# Toward Static Analysis of Immersive Attacks

Kadiray Karakaya*, Jonas Klauke*, Enes Yigitbas†
*Heinz Nixdorf Institute, Paderborn University
Email: {kadiray.karakaya, jonas.klauke}@upb.de
†Paderborn University
Email: enes.yigitbas@upb.de

***Abstract*—Immersive attacks are a novel class of security threats that emerge from the immersive nature of virtual reality (VR) interfaces. Unlike traditional cyber attacks that target users' sensitive information, immersive attacks target users' immersive experience: their visual perception and sense of direction. Despite their high damage potential, countermeasures for immersive attacks are still underexplored. In this work, we demonstrate how one can implement immersive attacks using OpenXR, a unifying standard that enables running vendor-independent VR applications on various VR platforms. We explore strategies for detecting such attacks through the perspective of static code analysis, a popular technique for application security vetting. We discuss the requirements and challenges for static analyses aimed at detecting immersive attacks, highlighting in particular the lack of cross-language support in existing tools and the absence of domain-specific knowledge needed to recognize these attacks.**

## I. Introduction

Virtual reality (VR) applications have witnessed widespread adoption across various domains. Their use cases range from entertainment and gaming [1] to education [2] and training simulations [3]. Their immersive nature enables intuitive user experiences with digital environments. Besides their wide use cases, VR applications also introduce a novel set of security challenges requiring special attention.

Like traditional software applications, VR applications can potentially compromise user privacy and security, posing risks such as data leaks and unauthorized access to sensitive information [4]. These privacy and security concerns highlight the importance of robust security measures in VR development and usage. Moreover, VR applications introduce a unique class of vulnerabilities known as *immersive attacks* [5], which target users' experiences within the immersive environments.

Compared to traditional attacks, the distinguishing feature of immersive attacks is that they directly target the users and their physical or mental well-being. Consumer-grade VR interfaces come with a set of measures to protect their users from physical harm. A common measure is a so-called virtual boundary (also referred to as a chaperone or guardian) that prevents users from running into walls or other objects in the real world [6]. *Chaperone attacks* [5] interfere with these virtual boundaries so that the users collide with real-world objects. With the same purpose, *human joystick attacks* [5] steer users to a target physical location. VR interfaces emulate real-world physics to provide a predictable immersive experience. Failing to do so causes confusion or notably visually induced motion sickness ( [7], [8]). *Disorientation attacks* [5] aim to confuse the users or make them dizzy. *Overlay attacks* [5] block users' view in the VR environment with unwanted visual content. Major VR device producers make third-party VR applications available through their application stores. This resembles mobile app stores with rather mature application security testing pipelines.

Static application security testing (SAST) is a well-established technique for detecting security vulnerabilities ( [9], [10]) in applications. Static analyses are often employed for practical security vetting of third-party applications published in application stores ( [11], [12]). As VR applications are distributed similarly through application stores, we believe SAST techniques will be increasingly in demand to cover VR-specific security issues, such as immersive attacks. Despite their potential benefits, the applicability of static program analysis to the VR application security domain is underexplored. This work aims to start a discussion around *static analysis of immersive attacks* by focusing on VR-specific requirements and the nature of these attacks. To understand such requirements, first, we implement two immersive attacks from the literature: frame drop attack [13] and view manipulation attack [5]. These attacks represent two different lines of work targeting users' immersive experiences. However, unlike the original implementations, we build these attacks on top of the OpenXR standard[1]. OpenXR defines a set of APIs to enable applications developed in different software development environments (typically game engines) to port to various VR systems. This, at the same time, also enables the attacks to be easily ported to various systems. Second, we research the existing works from the literature on applying static program analysis techniques to detect immersive attacks. We discuss that VR applications require tailored tooling regarding their build and runtime environments, application life-cycles, and frameworks. To summarize, this work presents: (1) two immersive attack implementations on top of OpenXR, and (2) the requirements and challenges for static analyses to detect these immersive attacks.

## II. Preliminaries

The immersive attacks implemented in this work build on the OpenXR standard. OpenXR defines vendor-independent APIs for developing extended reality (XR) applications (including virtual and augmented reality) that can run across different XR devices.

[1]https://www.khronos.org/openxr

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00069

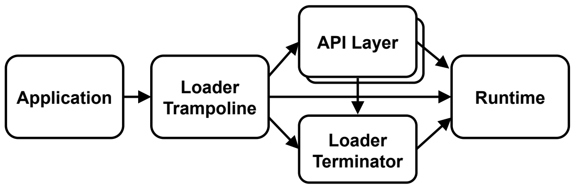

Fig. 1. OpenXR call chains

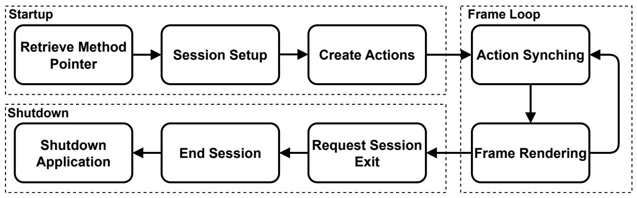

Fig. 2. Runtime method call order (simplified)

In OpenXR, a *loader* manages the connection between the *applications* and the *runtime*. The loader matches the API calls from the applications to their corresponding implementations in the runtime systems. OpenXR also enables sophisticated calls beyond the runtime system definitions through *API Layers*. Figure 1 shows the possible call chains that can occur within the OpenXR specification[2]. Method calls from applications are handled by the *Loader Trampoline*, which can directly lead them to the runtime. Alternatively, the calls can be directed to the API layers and the *Loader Terminator* before finally reaching the *Runtime*. Note that it is also possible that a method call is not forwarded to the runtime.

According to the OpenXR specification, to call a method, a pointer to the target method is retrieved via OpenXR's `xrGetInstanceProcAddr` method. It takes as input a reference to an instance and a target method name. Game engines, for instance, Unity, in this case, can hook into the method pointer retrieval process. Each OpenXR feature has a callback method called `HookGetInstanceProcAddr`. This method, by default, returns the native `xrGetInstanceProcAddr` method. However, by overriding the `HookGetInstanceProcAddr`, arbitrary methods can be returned instead of `xrGetInstanceProcAddr`.

```
1  [DllImport ("OpenXRIntercept", EntryPoint =
       "intercept_xrGetInstanceProcAddr")]
2  static extern IntPtr customInstanceProcAddr(IntPtr func);
3
4  override IntPtr HookGetInstanceProcAddr( IntPtr func){
5    return customInstanceProcAddr(func);
6  }
```

Listing 1. Returning a custom method by overwriting the callback method

In Listing 1, we override the callback method to return a custom method pointer in Line 4. To do so, we link to an external library's method, `customInstanceProcAddr` in Line 5. With this change, it is now possible to call an arbitrary proxy method for each method call occurring within the Unity application life cycle.

Figure 2 shows the simplified order of method calls[3]. The VR applications start with retrieving the required method pointers. Then the session starts, and application-specific actions are set up. Actions define controller event mappings, e.g., a button press. After the startup, the frame loop begins. It continuously synchronizes action events and renders the frame accordingly, until session exit is requested. Then the shutdown process is initiated. Proxying methods in different steps in the application life cycle enables different attack implementations.

## III. Immersive Attack Implementations

This section explains and demonstrates the frame drop and the view manipulation attack. We then evaluate the impact of these attacks through a user study.

### A. *Frame Drop Attack*

Reduced frame rates cause motion sickness and nausea in VR applications [7]. Based on this phenomenon, Odeleye et al. [13] demonstrated a frame-reduction attack by overloading the GPU. In this work, instead of overloading the GPU, we achieve the same effect by interfering with OpenXR methods in the application life cycle. In OpenXR, the rendering cycle is executed strictly in the following order: wait, begin, and end. By skipping the end phase of the current frame, the frame is discarded without disrupting the next rendering cycle. This enables frame drops simply by omitting the end phase. In OpenXR, the end phase is implemented by the `xrEndFrame` method. We implement the frame drop attack by replacing the `xrEndFrame` method with a custom version that does not always invoke the original `xrEndFrame` responsible for finalizing the rendering process. Listing 2 shows a simplified version of the attack, in which `xrEndFrame` skips frame rendering 30 times (Lines 4–6) before rendering a single frame by calling the original `xrEndFrame` method (Line 9).

```
1  static XrResult XRAPI_PTR intercepted_xrEndFrame(...)
2  {
3    if (s_xrEndFrame != nullptr) {
4      if (droppedFrames < 30) { //skip frame drawing
5        droppedFrames++;
6        return XR_SUCCESS;
7      } else { // draw the frame
8        droppedFrames = 0;
9        return xrEndFrame(...);
10     }
11   } else {
12     return XR_ERROR_RUNTIME_FAILURE;
13   }
14 }
```

Listing 2. simplified Frame Drop Attack

### B. *View Manipulation Attack*

As Casey et al. [5] demonstrated, manipulating a user's position and orientation in a virtual environment can disrupt their immersive experience. Their original work achieved this using a malicious configuration file to stage the attack. Our

[2]https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html
[3]https://docs.unity3d.com/Manual/ExecutionOrder.html

work achieves the same effect by interfering with OpenXR calls in the frame loop process. Within this loop, a user's head movements are captured as changes in the output of the `xrLocateViews` method, which informs the application of the HMD's position and orientation. In turn, the `xrEndFrame` method submits the corresponding data to render images accordingly. We exploit these methods to perform a view manipulation attack in two steps. First, we replace `xrLocateViews` with a malicious custom method. This method internally calls the original `xrLocateViews` but modifies the returned orientation and position values. Second, we replace `xrEndFrame` with our own method. One might expect the manipulated view to be rendered automatically; however, in OpenXR, an image is submitted with a fixed orientation and position, and will only render correctly if the user's current pose matches the pose associated with the image. Therefore, we must also update the orientation and position of the rendered image to reflect the manipulated view. A simplified version of this attack is shown in Listing 3. In this example, `xrLocateViews` modifies the position and orientation at Line 8, while `xrEndFrame` draws the frame using the original pose at Lines 14–15. In the listing, the term pose refers to the combination of position and orientation values.

```
1  intercepted_xrLocateViews(){
2    pose = xrLocateViews() //correct pose
3    saved_pose = pose
4    if (not initialized){ //setup the manipulation
5      changed_pose = pose
6      initialized = true
7    } else { //move the pose
8      pose = changed_pose - pose
9    }
10   return pose
11 }
12
13 intercepted_xrEndFrame(XrFrameEndInfo* frameEndInfo){
14   frameEndInfo.setPose(saved_pose) //use the correct pose
15   xrEndFrame(frameEndInfo) //render the frame
16 }
```

Listing 3. Simplified View Manipulation Attack

## C. Attack Impact

We have implemented a simple target application using the Unity game engine and OpenXR runtime system to demonstrate the attacks that were implemented. Figure 3 shows a screenshot from the target application. The user is positioned between two buckets: blue on the left and red on the right. A blue or red cube appears before the user, which can be grabbed with the hand-held controllers. The user's task is to sort the cubes by their colors into the right buckets. Once the cube is sorted, another one with a randomly chosen color spawns.

Using this target application, we have performed a small-scale user study (n=4) to evaluate the impact of the implemented immersive attacks. To this end, we have used the cyber sickness in VR questionnaire (CSQ-VR) by Kourtesis et al. [14]. It consists of six questions to assess the degree to which the users feel nausea, dizziness, disorientation, imbalance, fatigue, and discomfort. The questions are answered

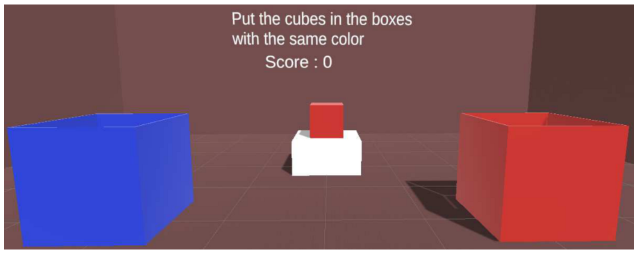

Fig. 3. Target application

on a 7-point Likert scale, where 1 means an absent feeling and 7 means an extreme feeling. Therefore, the score of a questionnaire may vary between 6 and 42. We have asked the participants to complete the task with three target application variants: with no attack, with the frame drop attack, and with the view manipulation attack. Each participant used all three variants for two minutes and then filled out the CSQ-VR questionnaire. Both attacks started when the participants reached a score of 5. We have configured the frame drop attack to reduce the frame rate to 20 frames per second instead of the default 120. The view manipulation attack inverted the head movement of the users, e.g., turning left made the view turn right. We introduced breaks between the variants to prevent the previous variants from affecting the results of the next variant.

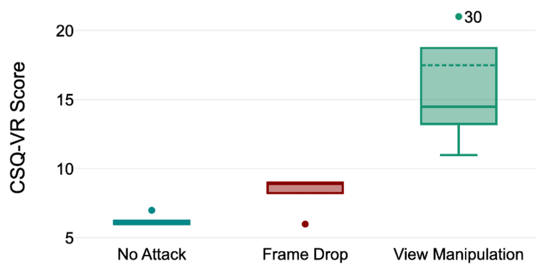

Fig. 4. Cybersickness felt by the users on each target application variant

Figure 4 shows the cybersickness levels the users feel on each target application variant. When no attack was present, only one of the participants felt very mild cybersickness. Under the frame drop attack, only one of the participants felt no cybersickness, while others reported mild feelings of cybersickness. Under the view manipulation attack, all participants reported cybersickness ranging from moderate to extreme. Our experiments show that the frame drop attack is less impactful than the view manipulation attack for causing cybersickness. However, the results might also depend on other factors, such as short and non-complex tasks.

We have demonstrated that leveraging the OpenXR features allows one to stage immersive attacks on VR applications. In line with the previous work ( [5], [13]), we report that immersive attacks interfere with users' immersive experience in VR environments, which could be harmful depending on the setting where these attacks occur.

## IV. Static Analysis of Immersive Attacks

The demonstrated immersive attacks occur at the application level by using OpenXR features. We assume three scenarios where these attacks can be executed. First, the attacker gets the victim to download and use the malicious application. Second, the attacker manipulates an existing application that uses a custom `getInstanceProcAddr` over a dynamic link library (DLL), by swapping out the DLLs in the application files. Third, the attacker provides a malicious OpenXRFeature with a useful function to developers, which may develop malicious apps without noticing or become the target themselves. Static application security testing can effectively detect such attacks on the application level.

Based on our observations on implementing the demonstrated attacks, we propose a set of static analyses that need to be implemented, as well as the technical challenges for the static analysis of immersive attacks.

### A. Required Static Analysis Implementations

Below, we present a set of static analyses that may help detect immersive attacks that we have presented in this work.

*1) Type Hierarchy Analysis:* The attacks that we have demonstrated rely on using OpenXR features. Therefore, the first step in the analysis is identifying whether OpenXR features are used in a target application. Static analysis tools can structurally check this by performing a *type hierarchy analysis*. Type hierarchy analyses [15] show the class inheritance relationships in a target program. By searching for classes that extend the `OpenXRFeature` class in the type hierarchy, one can reason about the existence of OpenXR features.

*2) Call Graph Analysis:* Since the attacks use custom OpenXR methods, we want to know whether a custom `getInstanceProcAddr` method is introduced. This enables returning pointers to malicious methods. This can be detected by checking whether the feature overwrites the default `hookGetInstanceProcAddr` method and returns a custom `getInstanceProcAddr`. This can statically be checked using a *call graph analysis*. Call graph analyses [16] not only reason about the existence of methods in a given application, but also, importantly, about whether they might *in fact* be invoked during the application runtime. After detecting a custom `getInstanceProcAddr` implementation, one needs to check whether it returns a valid OpenXR method. We identify two potential vulnerability sources. First, the parameters can be manipulated before calling the native `getInstanceProcAddr`, leading to a wrong method pointer being returned. Second, a custom method can be returned instead of the default one. In the first case, one can err on the safe side and soundly assume that the code is malicious. The returned custom method needs to be further analyzed in the second case, which we exploit to implement the presented attacks.

*3) Type-State Analysis:* The custom OpenXR method implementations must conform to the OpenXR specification. OpenXR defines a specific order in which each method needs to be called. For instance, we were able to implement the frame drop attack, explained in Section III-A, by not calling the expected end frame method. Therefore, the analysis has to check whether a custom method correctly forwards to a native or another custom method, which again needs subsequent analysis. Detecting such API misuse violations is well-studied in the static analysis domain [17]. One can formulate this as a *type-state analysis* [18] problem, where a finite-state automaton defines a set of states and the allowed transitions in between. The analysis then raises an error when a forbidden state is reached.

*4) Numeric Static Analysis:* We believe sophisticated attacks like immersive attacks require attack-specific analyses to be formulated. For instance, we implemented the view manipulation attack, explained in Section III-B, by manipulating the parameters and return values. Such manipulations are rather tricky for static analyses to detect precisely, as identifying such manipulations as malicious requires reasoning about runtime numeric values. One can formulate this as a *numeric static analysis* [19] problem, where runtime numeric values are approximated statically and unexpected value changes are identified as potentially malicious. On the other hand, at the cost of raising false positives, one can still soundly over-approximate detecting such attacks by simply marking all such value modifications as malicious.

### B. Static Analyses Challenges

In this exploratory study, we have observed that existing static analysis tools are not mature enough to detect immersive attacks. In the following, we explain some challenges that must be overcome.

*1) Cross-language Static Analysis:* The Immersive attacks presented in this work were implemented in Unity using the Windows operating system. In the following, we discuss the challenges faced in this environment; however, we expect them to carry over to other development environments. The implemented attacks utilize loading external DLLs, which requires static analyses to go beyond analyzing the C# language [20], the default language used in Unity, and be able to perform binary analysis of DLLs [21]. Moreover, Unity does not strictly use one language, as parts of the engine are written in C++. The development of VR applications is also not restricted to these two languages. For instance, the attacks analyzed by Vondráček et al. [22] exploit a weakness in a JavaScript library. The main challenge from a static code analysis perspective is supporting cross-language analysis or combining multiple language-specific analysis tools. To do so, one can also implement language-specific front ends to utilize a common intermediate representation. For instance, previous work [23] used the Soot [24] framework to analyze the CIL assembly code of the .NET Framework.

*2) Domain Knowledge:* Based on our experience in this study, we identify one of the key challenges to implementing static analyses for detecting immersive attacks as the lack of domain knowledge. To aid with this, we have implemented two existing immersive attacks from the literature. The static analysis approaches we discuss in this work are, therefore, only based on our experience in implementing these two attacks. We argue that given their potential impacts, immersive attacks need special attention. During this study, we observed that the nature of immersive attacks is still poorly understood. For instance, the Common Weakness Enumeration[4] (CWE), a database for known weaknesses that can be introduced by wrong programming, yields no results for a quick search on VR or immersive attacks. The CWE list contains ”Weaknesses in mobile applications”, but these also do not cover VR-specific vulnerabilities.

## V. Related Work

The security and privacy of VR and AR systems have been increasingly studied in the past few years [25]. Alismail et al. [26] give an overview of the cybersecurity threats of VR and AR systems in the literature. Adams et al. [27] describe the perception of users and developers regarding the security and privacy of VR systems. Guo et al. [28] present a security and privacy assessment tool called VR-SP detector for VR apps. VR-SP detector has integrated static analysis tools and privacy-policy analysis methods, but does not focus on immersive attacks. Similarly, in a recent work by Xu et al. [29], a novel framework to collect VR software security weaknesses from GitHub commit data is presented. Furthermore, Alghamdi et al. [30] introduce with xr-droid a benchmark dataset for AR/VR and security applications. While these approaches underline the importance of security and privacy aspects for VR and AR systems, they do not fully address the risk of immersive attack. In the following, we draw on prior work related to immersive attacks and their detection.

**Side-channel attacks.** VR systems are vulnerable to side-channel attacks. Arafat et al. [31] propose VR-Spy, a method to recognize keystrokes in VR environments through WiFi signal measuring to identify hand gestures. Ling et al. [32] demonstrate how keystrokes can be inferred by using a stereo camera or mobile sensor data. Rafique and Sen-Ching [33] show how one can jam or manipulate the tracking system of VR headsets by sending different light pulses via fake base stations. Odeleye et al. [13] demonstrated denial-of-service attacks by flooding the system with pings and overloading the GPU to cause frame drops. This work implements a similar frame drop attack using custom OpenXR features.

**Runtime and application level attacks.** VR systems are also vulnerable to attacks on the runtime systems or at the application level. Casey et al. [5] show how plain text configuration files used by SteamVR, combined with the OpenVR runtime, can stage various attacks. They coined the term immersive attacks. In this work, we implement the view manipulation attack, similar to the disorientation attack described by Casey et al. Vondráček et al. [22] introduce two novel attacks that target VR chat rooms. The ”man-in-the-room” attack enables an attacker to enter a virtual chat room and listen to the conversations without being noticed. The ”VR worm” attack enables a malicious program to spread through virtual chat rooms, activated via an unsanitized openUrl call. Static analyses detect such method calls, or even track, typically through a taint analysis, whether such calls can reach important application components.

**Attack detection.** Valluripally et al. [34] introduce an approach to identify attacks using machine learning to evaluate system anomalies. They [35] also formulate attack trees for risk evaluation of the VR applications regarding cybersickness. Qin and Hassan [36] present DyTRec, a tool for detecting certain method calls to support decisions for dynamic analysis. Molina et al. [37] present VRDepend, an analysis technique to determine dependencies between classes and scripts in Unity VR projects. Borelli et al. [38] present UnityLinter, a tool to detect code smells (e.g., performance-heavy methods) in Unity. The attack detection approaches in the literature either use dynamic analysis techniques or generic static analyses. It appears that sophisticated static analyses that can detect immersive attacks are nonexistent.

## VI. Future Directions

As discussed in Section IV-B, static analysis of immersive attacks requires both VR domain knowledge and extensions to the capabilities of existing static analysis tools. A helpful next step for the research community is to expand the available domain knowledge by systematically demonstrating a wider set of immersive attacks from the literature and documenting their implementation details. Since our preliminary experiments revealed that no existing static analysis tool can analyze immersive attacks out of the box, future work should focus on developing new tool sets, e.g., by building on existing frameworks ( [39], [40]) and supporting cross-language analysis. While immersive attacks are often considered in VR, they can equally target augmented reality (AR) applications. Therefore, a natural future direction is to investigate how static analysis approaches can generalize to the extended reality (XR) domain, encompassing both AR and VR. AR applications, in particular, require special attention because they often interact with the real world, raising distinct security concerns.

## VII. Conclusion

This work reports our experience building immersive attacks and motivates the need for novel static analyses to detect them. We have implemented two immersive attacks from the literature that exploit potential vulnerabilities in VR applications using the OpenXR standard. We have discussed how static analyses can detect immersive attacks and their limitations. Existing static analysis tools do not support detecting immersive attacks out of the box. We stress the need for domain expertise and a novel tool suite. Building such tooling requires the combined efforts of experts from the XR and the static analysis domains.

## Acknowledgments

We gratefully acknowledge Tobias Wilker’s contributions to this work.

[4]https://cwe.mitre.org

## References

[1] D. Checa and A. Bustillo, "A review of immersive virtual reality serious games to enhance learning and training," *Multimedia Tools and Applications*, vol. 79, no. 9, pp. 5501–5527, 2020.

[2] L. Freina and M. Ott, "A literature review on immersive virtual reality in education: state of the art and perspectives," in *The international scientific conference elearning and software for education*, vol. 1, no. 133, 2015, pp. 10–1007.

[3] E. Yigitbas, S. Krois, T. Renzelmann, and G. Engels, "Comparative evaluation of ar-based, vr-based, and traditional basic life support training," in *2022 IEEE 10th International Conference on Serious Games and Applications for Health(SeGAH), Sydney, Australia, August 10-12, 2022*. IEEE, 2022, pp. 1–8.

[4] S. Kulal, Z. Li, and X. Tian, "Security and privacy in virtual reality: A literature review," *Issues in Information Systems*, vol. 23, no. 2, pp. 185–192, 2022.

[5] P. Casey, I. Baggili, and A. Yarramreddy, "Immersive virtual reality attacks and the human joystick," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 2, pp. 550–562, 2021.

[6] J. Hartmann, C. Holz, E. Ofek, and A. D. Wilson, "Realitycheck: Blending virtual environments with situated physical reality," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, 2019, p. 1–12.

[7] H. K. Kim, J. Park, Y. Choi, and M. Choe, "Virtual reality sickness questionnaire (vrsq): Motion sickness measurement index in a virtual reality environment," *Applied ergonomics*, vol. 69, pp. 66–73, 2018.

[8] E. Chang, H. T. Kim, and B. Yoo, "Virtual reality sickness: a review of causes and measurements," *International Journal of Human–Computer Interaction*, vol. 36, no. 17, pp. 1658–1682, 2020.

[9] V. B. Livshits and M. S. Lam, "Finding security vulnerabilities in java applications with static analysis." in *USENIX security symposium*, vol. 14, 2005, pp. 18–18.

[10] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 6–pp.

[11] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 229–240.

[12] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," *ACM Transactions on Privacy and Security (TOPS)*, vol. 21, no. 3, pp. 1–32, 2018.

[13] B. Odeleye, G. Loukas, R. Heartfield, and F. Spyridonis, "Detecting framerate-oriented cyber attacks on user experience in virtual reality," 08 2021.

[14] P. Kourtesis, J. Linnell, R. Amir, F. Argelaguet, and S. E. MacPherson, "Cybersickness in virtual reality questionnaire (csq-vr): A validation and comparison against ssq and vrsq," *Virtual Worlds*, vol. 2, no. 1, pp. 16–35, 2023.

[15] A. Diwan, J. E. B. Moss, and K. S. McKinley, "Simple and effective analysis of statically-typed object-oriented programs," *ACM SIGPLAN Notices*, vol. 31, no. 10, pp. 292–305, 1996.

[16] M. W. Hall and K. Kennedy, "Efficient call graph analysis," *ACM Letters on Programming Languages and Systems (LOPLAS)*, vol. 1, no. 3, pp. 227–242, 1992.

[17] M. Schlichtig, S. Sassalla, K. Narasimhan, and E. Bodden, "Fum - a framework for api usage constraint and misuse classification," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 673–684.

[18] S. J. Fink, E. Yahav, N. Dor, G. Ramalingam, and E. Geay, "Effective typestate verification in the presence of aliasing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 17, no. 2, pp. 1–34, 2008.

[19] S. Wei, P. Mardziel, A. Ruef, J. S. Foster, and M. Hicks, "Evaluating design tradeoffs in numeric static analysis for java," in *European Symposium on Programming*. Springer International Publishing Cham, 2018, pp. 653–682.

[20] M. Belyaev, N. Shimchik, V. Ignatyev, and A. Belevantsev, "Comparative analysis of two approaches to static taint analysis," *Programming and Computer Software*, vol. 44, no. 6, pp. 459–466, 2018.

[21] J. Berkowitz and W. Qu, "A static over-approximate detection tool for at-risk dlls," in *Security and Management and Wireless Networks*, K. Daimi, H. R. Arabnia, and L. Deligiannidis, Eds. Cham: Springer Nature Switzerland, 2025, pp. 516–525.

[22] M. Vondráček, I. Baggili, P. Casey, and M. Mekni, "Rise of the metaverse's immersive virtual reality malware and the man-in-the-room attack & defenses," *Computers & Security*, vol. 127, p. 102923, 2023.

[23] S. Arzt, T. Kussmaul, and E. Bodden, "Towards cross-platform cross-language analysis with soot," in *Proceedings of the 5th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis*, 2016, pp. 1–6.

[24] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A java bytecode optimization framework," in *CASCON First Decade High Impact Papers*, 2010, pp. 214–224.

[25] F. Roesner and T. Kohno, "Security and privacy in the metaverse," *IEEE Security & Privacy*, vol. 22, no. 1, pp. 7–9, 2024.

[26] A. Alismail, E. Altulaihan, M. H. Rahman, and A. Sufian, "A systematic literature review on cybersecurity threats of virtual reality (vr) and augmented reality (ar)," *Data Intelligence and Cognitive Informatics: Proceedings of ICDICI 2022*, pp. 761–774, 2022.

[27] D. Adams, A. Bah, C. Barwulor, N. Musaby, K. Pitkin, and E. M. Redmiles, "Ethics emerging: the story of privacy and security perceptions in virtual reality," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 2018, pp. 427–442.

[28] H. Guo, H.-N. Dai, X. Luo, G. Xu, F. He, and Z. Zheng, "An empirical study on meta virtual reality applications: Security and privacy perspectives," *IEEE Transactions on Software Engineering*, 2025.

[29] Y. Xu, J. Chen, Z. Qi, H. Chen, J. Wang, P. Hu, F. Liu, and S. He, "An empirical study on virtual reality software security weaknesses," *arXiv preprint arXiv:2507.17324*, 2025.

[30] A. Alghamdi, A. Al Kinoon, A. Alghuried, and D. Mohaisen, "xr-droid: A benchmark dataset for ar/vr and security applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 2, pp. 1418–1430, 2024.

[31] A. Al Arafat, Z. Guo, and A. Awad, "Vr-spy: A side-channel attack on virtual key-logging in vr headsets," in *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*. IEEE, 2021, pp. 564–572.

[32] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, and X. Fu, "I know what you enter on gear vr," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 241–249.

[33] M. U. Rafique and S. C. Sen-ching, "Tracking attacks on virtual reality systems," *IEEE Consumer Electronics Magazine*, vol. 9, no. 2, pp. 41–46, 2020.

[34] S. Valluripally, B. Frailey, B. Kruse, B. Palipatana, R. Oruche, A. Gulhane, K. A. Hoque, and P. Calyam, "Detection of security and privacy attacks disrupting user immersive experience in virtual reality learning environments," *IEEE Transactions on Services Computing*, 2022.

[35] S. Valluripally, A. Gulhane, K. A. Hoque, and P. Calyam, "Modeling and defense of social virtual reality attacks inducing cybersickness," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4127–4144, 2021.

[36] X. Qin and F. Hassan, "Dytrec: A dynamic testing recommendation tool for unity-based virtual reality software," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[37] J. Molina, X. Qin, and X. Wang, "Automatic extraction of code dependency in virtual reality software," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 381–385.

[38] A. Borrelli, V. Nardone, G. A. Di Lucca, G. Canfora, and M. Di Penta, "Detecting video game-specific bad smells in unity projects," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 198–208.

[39] G. Teixeira, J. Bispo, and F. F. Correia, "Multi-language static code analysis on the lara framework," in *Proceedings of the 10th ACM SIGPLAN international workshop on the state of the art in program analysis*, 2021, pp. 31–36.

[40] K. Karakaya, S. Schott, J. Klauke, E. Bodden, M. Schmidt, L. Luo, and D. He, "Sootup: A redesign of the soot static analysis framework," in *Tools and Algorithms for the Construction and Analysis of Systems*, B. Finkbeiner and L. Kovács, Eds. Cham: Springer Nature Switzerland, 2024, pp. 229–247.