Principled and Automated Approach for Investigating AR/VR Attacks

Muhammad Shoaib
University of Virginia

Alex Suh
University of Virginia

Wajih Ul Hassan
University of Virginia

Abstract

As Augmented and Virtual Reality (AR/VR) adoption grows across sectors, auditing systems are needed to enable provenance analysis of AR/VR attacks. However, traditional auditing systems often generate inaccurate and incomplete provenance graphs, or fail to work due to operational restrictions in AR/VR devices. This paper presents REALITYCHECK, a provenance-based auditing system designed to support accurate root cause analysis and impact assessments of complex AR/VR attacks. Our system first enhances the W3C PROV data model with additional ontology to capture AR/VR-specific entities and causal relationships. Then, we employ a novel adaptation of natural language processing and feature-based log correlation techniques to transparently extract entities and relationships from dispersed, unstructured AR/VR logs into provenance graphs. Finally, we introduce an AR/VR-aware execution partitioning technique to filter out forensically irrelevant data and false causal relationships from these provenance graphs, improving analysis accuracy and investigation speed. We built a REALITYCHECK prototype for Meta Quest 2 and evaluated it against 25 real-world AR/VR attacks. The results show that REALITYCHECK generates accurate provenance graphs for all AR/VR attacks and incurs low runtime overhead across benchmarked applications. Notably, our execution partitioning approach drastically reduces the size of the graph without sacrificing essential investigation details. Our system operates non-intrusively, requires no additional installation, and is generalizable across various AR/VR devices.

1 Introduction

The rising popularity of Virtual and Augmented Reality (AR/VR) technologies across various sectors introduces unique cybersecurity challenges and risks, including Advanced Persistent Threats (APTs) [97, 99]. Adversaries exploit AR/VR headsets to manipulate perception [103], steal sensitive data [98, 99], and alter device configurations to distort visual experiences [55, 56]. This exploitation underscores the need for a specialized auditing system capable of conducting causal analysis, such as pinpointing root causes and assessing the impacts of unique AR/VR attacks. Recognizing the inherent cybersecurity risks of AR/VR's immersive nature, regulations and policymakers [44] also emphasize the urgent need for auditing systems to safeguard their potential.

Data provenance provides a comprehensive account of data entity transformations, making it an ideal technique for auditing AR/VR devices. In a provenance-aware system, audit logs (also known as system logs) are parsed into a provenance graph that encapsulates the entire history of the system's operation [69, 76, 80, 87, 90, 105]. In this graph, vertices represent system entities (e.g., files and processes) while the edges reflect causal relationships between these entities. Given an attack symptom (threat alert), users can leverage this graph to perform causal analysis.

Unfortunately, traditional android auditing systems [63, 106, 108, 111] and AR/VR network logging framework [102] struggle to perform accurate causal analysis of AR/VR attacks as outlined in Table 1. Existing methods require root access to log syscalls, which is increasingly infeasible for most AR/VR devices as Android diminishes the provision of root [26]. Dependent on the `strace` utility and system file modifications, these methods are ineffective without root access. This challenge arises from the closed-system architectures of AR/VR devices; unlike Android phones where syscall tracing is widely available [4], AR/VR platforms restrict privileged access to syscalls and employ customized security measures that hinder the direct use of traditional mobile auditing methods.

Second, as existing systems rely on system-layer audit logs (e.g., syscalls), they overlook critical AR/VR-specific attributes prevalent in higher layers of the AR/VR software stack, such as spatial boundaries, head movement data, and perceptual manipulations. Incorporating these attributes into provenance graphs is pivotal for understanding subtle attack vectors. For instance, Casey et al. highlight the significance of understanding unauthorized modifications to users' virtual environments [55]. While OVRseen [102] audits the Ocu-

Table 1: Limitations of existing systems in investigating AR/VR attacks. DF. stands for DroidForensics.

Properties	Dagger [108]	DF. [111]	AppAudit [106]	OVRSeen [102]	Our System
Runs without root access	✗	✗	✓*	✓*	✓
Supports closed-source apps	✓	✓	✗	✓	✓
Integrates unique AR/VR attributes	✗	✗	✗	✗	✓
Multi-layer causal analysis	✗	✗	✗	✗	✓
AR/VR-aware execution partitioning	✗	✗	✗	✗	✓

*Demands complex dynamic analysis: infeasible on end-user devices.

lus VR SDK, its focus is limited to network-layer audit logs, leading to incomplete attack reconstructions. Furthermore, OVRseen requires dynamic analysis expertise and targets the now-deprecated OVR SDK from Meta.

Third, another key challenge in auditing AR/VR systems is the lack of publicly available labeled datasets for AR/VR-specific entities. This absence hinders the accurate identification and correlation of AR/VR-specific entities, such as spatial boundaries, virtual objects, controller feedback, and endpoint transfers within the unstructured logs generated by AR/VR systems. Additionally, the lack of mechanisms to correlate logs across different layers of AR/VR further complicates tracking AR/VR entities and their interactions.

Finally, existing auditing systems suffer from the *dependence explosion* problem [76, 80]. For a long-running process, an output event (e.g., creating a malicious file) is assumed to be causally related to all the preceding input events (e.g., network receive). This conservative assumption creates false causal relations and large provenance graphs. Although researchers have put forward execution partitioning techniques as solutions [78, 80, 86], these techniques fail to address the unique execution patterns of AR/VR applications. Such techniques target Windows/Linux hosts and rely on traditional event-handing loops often absent in AR/VR applications [34].

To address the aforementioned limitations, we introduce RealityCheck, a provenance-based auditing system that empowers investigators to effectively analyze the root causes and impacts of AR/VR attacks. Our system is informed by a comprehensive study of AR/VR attacks [1, 13, 21, 55, 56, 83, 88, 91, 98, 100, 101, 103, 104, 113]. These attacks are meticulously crafted and published in top-tier academic conferences and journals, revealing vulnerabilities and threat models critical to current AR/VR systems. *To the best of our knowledge, our work is the first in the literature* to systematically categorize AR/VR attacks as per the MITRE ATT&CK [18] framework and offer a comprehensive auditing system for AR/VR ecosystem. The attacks and the corresponding proposed Tactics, Techniques, and Procedures (TTPs) are summarized in Tables 2 and 9, respectively. Further details on the novelty of our attack categorization are discussed in §8.

Our Contributions.

Attack Modelling. First, through studying existing attacks on hybrid AR/VR devices[1], we identified eight AR/VR-specific attributes that are necessary to understand AR/VR attack behaviors. With a comprehensive understanding of these attributes, we augmented the W3C PROV-DM specification [49] by encoding additional ontology. This enhancement enables us to construct provenance graphs that capture pivotal AR/VR entities and causal relationships.

Log Analysis & MPG Generation. Building on our AR/VR-centric PROV-DM model, we first collect and unify logs across the AR/VR software stack (application, SDK, platform, endpoint), even when they lack a consistent schema or explicit identifiers. To handle the inherent unstructured nature of these logs, RealityCheck introduces a specialized NLP pipeline – combining Named Entity Recognition (NER) and Part-of-Speech (POS) tagging – that recognizes and labels domain-specific AR/VR entities, along with their *who-did-what* relationships. This domain-adaptive parsing is key to capturing higher-level AR/VR events (e.g., forced boundary adjustments or rapid flicker toggles) that traditional audit systems miss. Further, our custom feature selection method extracts and correlates logs across the various layers of the AR/VR software stack, resulting in the creation of a Multi-layer Provenance Graph (MPG).

AR/VR-Aware EP. Third, to mitigate the dependence explosion in AR/VR applications, we implemented a novel Execution Partitioning (EP) technique based on the OpenXR application lifecycle model [31]. Our key insight is to leverage OpenXR's session state transitions and event polling mechanism to accurately partition the application execution and user interactions into discrete execution units. By retaining causal dependencies strictly within these units, we eliminate false dependencies in AR/VR provenance graphs.

MPG Pruning. Fourth, we design an automated graph pruning mechanism to further improve the accuracy of graphs generated by RealityCheck. Our pruning strategy is based on Direct Forensic Connection (DFC) and Indirect Forensic Connection (IFC) principles, which identify and remove forensically irrelevant vertices and edges from the AR/VR provenance graph.

We implemented a RealityCheck prototype for Meta Quest 2, which holds over 90% of the market share [15], and thoroughly evaluated its efficacy and performance against a set of 25 known AR/VR attacks detailed in Table 2. Notably, we injected malicious code into real AR/VR applications (from the Quest and SideQuest stores) to replicate attacker tactics. We then augmented these tests with an APT-style approach using MITRE Caldera, demonstrating full attack chains that commence with phishing on a Windows endpoint and proceed to malicious sideloading on the headset. These attacks represent real-world scenarios; for instance, Luo et al. propose an eavesdropping attack [83] and Cheng et al. introduce immersive visual deception attacks [57], which exemplify the type of real-world threats included in our evalua-

[1] Hybrid AR/VR devices operate independently but provide additional features when connected to endpoints like a Windows PC.

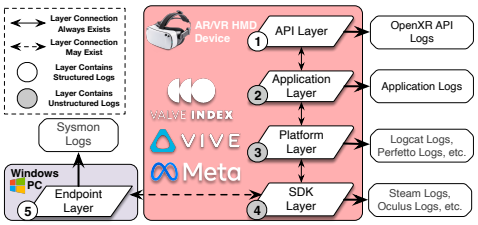


Figure 1: The AR/VR ecosystem comprises five system layers.

tion of REALITYCHECK. An in-depth analysis of the criteria used to select the attacks and applications, underscoring their applicability to real-world scenarios, is provided in §9.

Our results show that REALITYCHECK-generated provenance graphs enable more accurate AR/VR attack investigations than OVRseen [102], effectively capturing the root cause and impacts of AR/VR attacks. REALITYCHECK incurs minimal runtime overhead of less than 6% and uses lifecycle-based partitioning and graph pruning techniques, reducing graph size by up to 76% without losing critical investigation details. The average query times for impact analysis is 2.25 secs per attack, and the average query times for root cause analysis is 2.12 secs. Additionally, through an ablation study of our NLP components, we confirm that combining our POS tagging and fuzzy entity resolution significantly boosts precision and recall (up to 98%), while also reducing extraneous graph data. We validate the system's efficacy through two AR/VR attack case studies, highlighting its capacity for precise causal analysis (§3, §7). Further, our system is generalizable to other AR/VR devices like HTC Vive XR Elite [47] and adaptable to emerging AR/VR threats (§B.2, §B.1).

Availability. Our code and data is available at https://anonymous.4open.science/r/RealityCheck-Paper168/.

2 AR/VR Attack Modelling & Logging

AR/VR technologies introduce unique cybersecurity risks, as manipulating immersive content can lead to data theft, physical harm, or psychological distress. Traditional security attributes, like file or network activity, cannot capture these multidimensional threats. For example, altering virtual boundaries exploits system vulnerabilities and user perception, potentially causing physical collisions or confusion. Modeling AR/VR attacks requires analyzing AR/VR-specific attributes rooted in device software, sensory pipelines, and virtual environment design. We identified eight novel attributes encapsulating AR/VR threats. These attacks are detailed in Table 2, with selection methodology in Appendix F and systematization following MITRE TTPs [18] in Appendix E.

A1 User Gesture/Input Manipulation: Adversaries can interfere with user actions, such as intercepting or falsifying controller inputs or hand gestures [55]. *Example:* A malicious app invisibly overlays a button to trick the user into revealing sensitive information.

Table 2: AR/VR attacks overview. Layer numbers in Figure 1 show potential (bracketed) and definite (non-bracketed) manifestations. See Table 9 in Appendix E for attack procedures and artifacts/tools that were used to simulate the attacks. The last row provides references for the real open-source apps used to inject the payloads and run attacks. Disc.: discovery, VE: Virtual Environment, HMD: head-mounted (AR/VR) device.

No.	Attack Name	Unique Attributes	Layers	Real Apps.
1	Overlay partial screen [103]	A3, A4	1, 2, 3, 4, (5)	[50, 51]
2	Overlay entire screen [103]			
3	Object-in-the-middle attack [57]			
4	Object erasure attack [57]			
5	Alter VE coordinates [55]	A1, A3, A4		
6	Alter VE sensitivity [55]			
7	Software disc. [13]	A6, A8	1, (2), 3, 4, (5)	[2, 11]
8	System info. disc. [13]			
9	System service disc. [13]			
10	Network config. disc. [89]			
11	Foreground data access [83, 99]	A1, A2, A7, A8	1, (2), 3, 4, 5	[27, 39]
12	Immersive browsing hijack [37]	A1, A3, A6		
13	Collect clipboard data [89]	A2, A6, A7		
14	Automated exfiltration [13, 37]	A1, A2	1, (2), 3, 4	
15	Exfiltration via endpoint [91]	A6	1, (2), 3, 5	
16	Cyber-sickness: dizziness [104]	A3, A5	1, (2), 3, 4, (5)	[7, 40]
17	Force play audio [29]			
18	Cyber-sickness: epilepsy [104]			
19	Input capture [83]	A1, A2, A6		
20	Credential access [57]			
21	File deletion [71, 89]	A6		
22	Delete user data [89]			
23	Service stop [29]			
24	Account access removal [29]			
25	HMD shutdown [37, 71]	A3, A6	1, (2), 3, 4, 5	[48]

A2 Head Movement Data Access/Manipulation: Unauthorized monitoring or alteration of *head tracking data* [91, 98, 99]. In AR/VR, the head pose is integral to user perspective, so capturing subtle changes can reveal private information like passcodes typed via head rotation.

A3 Perceptual/Visual Deception: Injecting deceptive visuals, illusions, or overlays that alter the user's real-time perception of the virtual environment (VE) [56]. *Example:* The "Object-In-The-Middle" (OITM) attack [57] places an invisible collider over a password entry field, stealing keystrokes undetected.

A4 Spatial Boundary Manipulation: Changing *virtual boundaries* or "Guardian" limits to mislead or disorient the user [55, 91, 103]. One example is the "Chaperone Attack," which quietly shifts walls in the virtual environment so the user walks into physical obstacles.

A5 Physical Harm: Triggering motion sickness, nausea, or even epileptic seizures by exploiting the immersive sensorial feedback of AR/VR devices [55, 103, 104]. *Example:* Rapidly flashing lights or forcibly generating intense haptic signals can cause physiological discomfort.

A6 Immersive Session Integrity Compromise via Endpoint Access: Gaining unauthorized entry to the VR session by targeting an external endpoint (often a tethered Windows PC) [91]. For instance, an attacker compromises a user's PC and then deploys malicious payloads onto the headset to hijack its immersive session.

A7 6DoF Data Access/Manipulation: Extracting or modify-

ing *six degrees of freedom* (6DoF) tracking data (i.e., x, y, z positions and yaw, pitch, roll) [98, 99]. If stolen, it may reveal user movement patterns or be used for biometric inferences.

A8 Background App Access Compromise: Exploiting background processes in AR/VR OS environments to snoop on user data or manipulate running apps. This phenomenon is rarer in standard Android but becomes critical in AR/VR, where background apps can still track real-time sensor streams [113].

Our motivation for modeling attacks is to understand potential threats on AR/VR devices, guide our log collection strategy, and evaluate REALITYCHECK against these attacks. This aids in optimizing our data collection and maximizing its research relevance. We conducted a comprehensive study of academic papers, CVE databases, and relevant GitHub repositories from the past five years, using relevant keywords (detailed in Appendix F). This study yielded 25 distinct attacks targeting various aspects of major hybrid AR/VR devices [15] such as the Meta Quest (2 and Pro) [20, 22], HTC Vive [47], and the Valve Index [45]. Table 2 lists the identified attacks, their unique attributes, and the likely impacted AR/VR system layers. This attack model forms the basis of our AR/VR auditing system. As outlined in Table 2, each attack targets specific AR/VR device layers. For instance, the OITM attack (#3) uses A3 (Perceptual/Visual Deception) and A4 (Spatial Boundary Manipulation), whereas the Chaperone Attack (#5, #6) manipulates VE coordinates and VE sensitivity, emphasizing the cognitive dimension (A1, A3, A4). Some attacks (e.g., #16 to #18) aim to cause dizziness or epilepsy in the user by exploiting the immersive nature of AR/VR (A3, A5). Meanwhile, attacks like #7 to #9 focus on endpoint-based entry (A6, A8), reflecting a multi-layer infiltration approach.

Figure 1 illustrates how AR/VR devices typically involve five layers: (1) platform layer, which provides system event data like method calls and stack traces, (2) SDK layer, showcasing device-specific events, (3) API layer, capturing API function invocations, (4) application layer, where applications run, and (5) endpoint layer. Attacks may manifest in one or more of these layers. For example, a device driver compromise at the SDK layer may grant direct access to raw motion sensor data (A2, A7) while an endpoint layer exploit may allow attackers to push malicious apps into a closed HMD system (A6).

In §5, we identify six main log sources for tracing AR/VR attacks: Logcat logs, Perfetto logs, Application logs, Device logs, Windows Event logs, and OpenXR API logs. Logcat logs [14] offer a system overview, Perfetto [36] logs provide common keys for system-app interactions, Application logs monitor user actions, Device logs [30, 46] outline AR/VR device-PC interactions, Windows Event logs [43] record endpoint activities, and OpenXR API logs [31] detail chronicle unique application lifecycle events. These logs are crucial for lifecycle-based execution partitioning and app state identification as discussed in §5.4. Users can access this information *without installing new tools* on their AR/VR device as logs can be collected via the existing Android toolkit, making our approach accessible.

3 Motivating AR/VR Attack Example

Attack Scenario. In the depicted scenario, an attacker exploits a Windows PC, to drop a malicious payload on an AR/VR headset. One potential outcome is a Human Joystick Attack, where the attacker quietly alters the headset's spatial boundaries (the invisible "walls" defining the user's safe area). As a result, the user may be misled into moving dangerously close to physical objects, risking collision or injury. Meanwhile, the attacker also eavesdrops on user inputs (e.g., gesture or controller data) to steal sensitive information. In this attack scenario, the underlying intrusion detection system raises an alert upon detecting an unusual boundary collision.

Existing Tools' Limitations. In Figure 2(a), we see an AR/VR provenance graph that is incomplete. Although existing forensic tools can reconstruct the Windows side of the attack (e.g., how the PC was breached), they fail to cover AR/VR-specific events such as altering a spatial boundary. The missing connections stem from AR/VR logs being unstructured, incomplete, and lacking crucial causal details. Identifying how these logs should be collected at each layer of the AR/VR software stack (provided in §2) is also non-trivial, which contributes to the fragmented view of the attack.

Intermediate Provenance Graph. Figure 2(b) shows an intermediate step, where we apply AR/VR-focused techniques—our provenance model (§5), NLP for unstructured logs (§5.1), and multi-layer feature correlation (§5.2). The resulting graph covers both the Windows PC and the headset, but is still very large and cluttered with irrelevant details, suffering from the dependency explosion problem and from benign activities and system maintanance logs, a unique challenge we observed when using off-the-shelf logs, not handled by existing log garbage collection tools [68, 79]. We further explain why existing tools to solve the dependency explosion problem are not applicable to the novel AR/VR execution model [31] in §5.4.

Investigation with REALITYCHECK. Figure 2(c) shows the refined provenance graph produced by REALITYCHECK after integrating lifecycle-based partitioning (§5.4) and graph pruning (§5.5). These steps remove spurious dependencies and extra logs, yielding a concise attack narrative: (1) the attacker compromises a Windows PC (1–80), (2) modifies AR/VR boundaries (81–95), and (3) eavesdrops on user inputs (104–123). When the intrusion detection system flags a suspicious boundary collision (edge 103), analysts can backtrack using REALITYCHECK to trace the root cause back to IP `172.20.16.93`, where `mal.exe` was downloaded

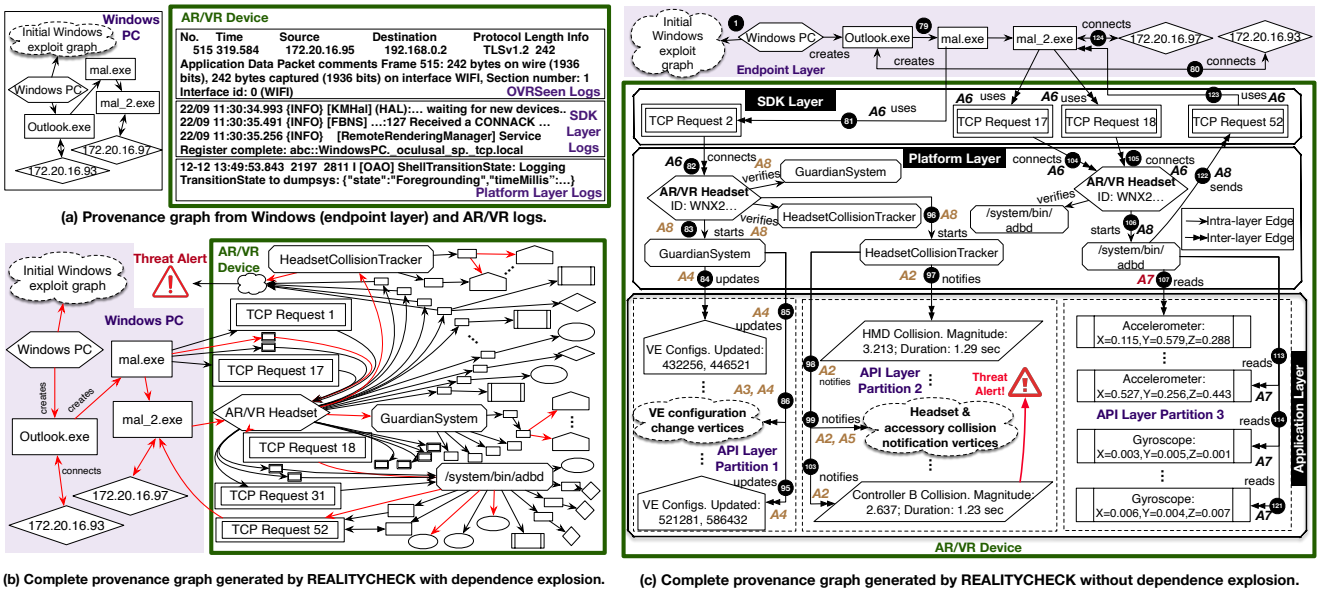


Figure 2: Provenance graphs generated using traditional auditing systems vs. REALITYCHECK. We exclude the initial Windows PC exploit for readability purposes. The "Windows PC" graph in (a) is obtained using Sysmon, the AR/VR headset logs are provided to emphasize their unstructured nature. In (b), the naive approach (without REALITYCHECK's lifecycle-based execution partitioning) lumps all prior events into the same causal chain, causing false dependencies (denoted as black edges for illustration only). This volume of spurious links conceals the true malicious path (red edges), hindering analysts from pinpointing the attack sequence. The graph in (c) fixes the dependency explosion and shows a threat alert initiated by an HMD collision within guardian boundaries; backtracking leads to a malicious program `mal.exe` on the Windows PC, pinpointing the attack's origin for the investigator. API layer partitions are highlighted by dotted squares. Attributes highlighted in brown are used to denote the parent vertices, child vertices and edges that are unique to AR/VR.

(**Immersive Session Integrity Compromise via Endpoint Access**). This payload triggers additional AR/VR reconfigurations (81–95) demonstrating **Spatial Boundary Manipulation**. Meanwhile, `HeadsetCollisionTracker` logs collisions (97–103), and edges 104–123 reveal a background eavesdropping attack [98, 99] via `adbd` (**Background App Access Compromise**). Accessing accelerometer/gyroscope data indicates **Head Movement Data Access/Manipulation** and **6DoF No Permission Sensory Data Access/Manipulation**, enabling key-logging by tracking head movements and leaking sensitive information to IP `172.20.16.97`. Overall, this example illustrates the range of AR/VR-specific threats—from perceptual deception to boundary manipulation—that conventional auditing systems overlook, underscoring the need for specialized AR/VR-focused auditing.

4 Threat Model and Assumptions

REALITYCHECK is designed to enable security analysts to recover attack stories within the AR/VR ecosystem. In our threat model, we consider attacks that either directly target the AR/VR devices, or indirectly target them through a connected Windows PC[2]. Attackers could employ various methods such as exploiting software vulnerabilities or using social engineering tactics to deceive users into installing malicious apps on their AR/VR headsets. These attacks can lead to numerous threats, ranging from discovering sensitive system information and data exfiltration to compromising the device to manipulate the user's perception through visual or auditory alterations. Furthermore, it's important to clarify that not all attacks originate from the AR/VR device alone. An attacker may compromise a Windows PC through techniques such as privilege escalation or exploiting system vulnerabilities. With such access, an attacker could use Android toolkits like Fastboot and ADB to launch complex attacks on the AR/VR device connected to the PC via USB or TCP connections.

We exclude attackers who have obtained root access to devices and who exploit vulnerabilities in AR/VR hardware, as we assume device integrity. Securing the device is a problem orthogonal to our work. This assumption aligns with related prior work [102]. Additionally, we make several assumptions for REALITYCHECK in line with existing data provenance works [69, 76, 80, 87, 90, 105]. We assume the logging system components are part of a trusted computing base, ensuring the integrity of logs collected at various vantage points. Additionally, we assume attacks commence only after our system begins monitoring victim devices and that collected logs remain uncompromised. To safeguard log integrity, existing tamper-evident logging solutions can be employed [52, 90].

[2]AR/VR devices primarily use Windows PCs (see §5.2). With Linux unsupported, we target Windows OS for our endpoint layer.

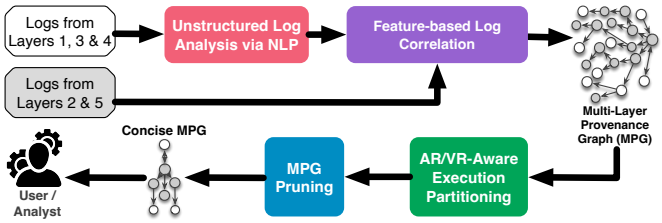


Figure 3: Workflow of REALITYCHECK.

5 Design

Our system leverages *OpenXR*–the unifying API for AR/VR platforms–to achieve device-agnostic development, cross-platform compatibility, and seamless app integration [33]. Because major arvr manufacturers are converging on OpenXR for core runtime functionalities, REALITYCHECK inherits a naturally broad scope, offering adaptability across heterogeneous headsets. Furthermore, most standalone AR/VR devices, including the Meta Quest 2 and HTC Vive XR Elite, run customized Android distributions, so our platform-layer logs (e.g., Perfetto logs, Android logging) are present across devices. Meanwhile, SDK logs from Oculus (Quest) [30] or SteamVR (Vive/Index) [46], together with standard app logs [5], ensure REALITYCHECK can capture forensically relevant AR/VR events without device-specific root-level access. As demonstrated in §B.2, this framework generalizes across diverse headsets and software stacks.

Although REALITYCHECK bases its provenance graph on the W3C PROV-DM specification [49], *AR/VR attacks require additional concepts* to capture spatial boundaries, continuous sensor data, and immersive user interactions. Figure 4 shows our AR/VR-centric ontology overlaying the standard PROV model: we introduce AR/VR-specific entities (e.g., *SpatialBoundary*, *GuardianSystem*) on top of traditional system vertices (e.g., *Process*, *File*), along with new relationships e.g., those describing user sensory data, spatial boundary manipulation. This structured graph allows REALITYCHECK to trace both conventional system activity *and* unique AR/VR events within a single, coherent provenance view, enhancing attack investigations within the AR/VR ecosystem.

5.1 Unstructured Log Analysis

Unstructured logs from Platform, Application, and SDK layers (§2), each with unique textual layouts, pose significant challenges for extracting entities and relationships for provenance-graph construction. Owing to the closed-source nature of AR/VR devices like the Meta Quest 2 and HTC Vive XR Elite [3], simple regex-based log parsing is inadequate. Instead, Algorithm 1 details our NLP pipeline, which leverages named-entity recognition (NER), part-of-speech (POS) tagging, and dependency parsing to *automate* the extraction of AR/VR-relevant entities and edges.

Overall Pipeline and Motivation. Algorithm 1 proceeds

	AR/VR Entity Legend	Unique AR/VR Attributes	Relationship Legend
E1	Application (foreground), Application (background)	A1, A2, A8	Runs (E11) Generates (E4) Interacts (E6)
E2	Endpoint Transfer	A6	Establishes (E11) Transmits (E5, E9)
E3	Virtual Environment Setup	A3, A4	Alters (E1) Contains (E6)
E4	AR/VR Abnormalty Notification	A5	Generates (E1)
E5	Sensory Data	A1, A2, A7	Accesses (E1) Transmits (E2)
E6	Virtual Object or UI Element	A3	Creates/Alters (E1) Resides (E3)
E7	Haptic Feedback	A5	Generates (E1)

E8 Process E9 File E10 Socket E11 Device

Figure 4: Novel classification of AR/VR specific vertices and edges in the provenance graphs generated by REALITYCHECK.

in *two stages*: (1) *Training*, where we fine-tune a custom NER model on an annotated AR/VR log dataset, and (2) *Runtime*, where we apply the model to new, unstructured logs. Our main motivation is that AR/VR logs (e.g., from `Logcat`, `Oculus logs`, or `Perfetto`) lack standardized tokens or a fixed schema; even the same device can generate logs with different fields or formats across OS updates. Hence, a robust NLP-based approach can learn from user-labeled domain examples, generalize to previously unseen logs, and *automatically* infer relationships (e.g., *did the device 'create'/'read'/'connect' to some resource?*) without requiring root access.

Step 1: Dataset and Labeling. Our dataset spans 25 real-world AR/VR applications, with a total of 100 hours of usage logs. AR/VR logs lack publicly-available labeled corpora or ontologies, we therefore annotated logs using the ontology of AR/VR-specific entities (e.g., `GuardianSystem`, `SpatialBoundary`) from Figure 4. Each of the 11 entities in our system was labeled across about 5 logs, yielding a balanced dataset. We separated the training logs from those used in actual attacks.

Step 2: Text Preprocessing. We begin by tokenizing each log entry (e.g., `["GuardianSystem:","HMD","collision","Detected"]`), then apply POS tagging to assign grammatical roles (`HMD` and `collision` as nouns, `Detected` as a verb), and finally use lemmatization to reduce variants (`"Detected"` → `"Detect"`). By mapping text to subject-verb-object relationships, we can accurately identify entities and their interactions. For instance, "`GuardianSystem: HMD collision Detected`" reveals that `GuardianSystem` is the subject (an *Agent*) performing the *Detect* action on the `HMD collision` object (an *Entity*).

Step 3: NER Model (CNN-based) and Customization. We fine-tuned the spaCy `en_core_web_sm` model, which employs a CNN under the hood. To adapt it for our AR/VR ontology, we introduced custom entity labels (e.g., `SpatialBoundary`) and trained for 200 epochs using a com-

Algorithm 1: NLP PIPELINE

```
   /* (Step 1) Train a domain-specific NER model on annotated logs */
1  Training Phase: Input: Annotated Logs; Output: Custom NER Model.
2  Model ← FINETUNENERMODEL(training_dataset)
3  SAVEMODEL(Model)
   /* (Step 2) Parse logs at runtime to build the provenance graph */
4  Runtime Phase: Input: Unstructured Logs; Output: Vertices, Edges.
5  Model ← LOADMODEL(); Vertices, Edges ← ∅,∅
6  foreach log in logs do
       /* (Step 2a) Text preprocessing: tokenize, POS-tag, etc. */
7      Tokens, POS, Deps, Lemmas ← SPACYPREPROCESS(log)
8      FilteredTokens ← FILTERTOKENS(Lemmas)
       /* (Step 3) Named Entity Extraction & Fuzzy Matching (if any) */
9      Entities ← ExtractEntities (FilteredTokens)
10     foreach entity in Entities do
11         entity ← RESOLVEENTITY(entity, Entities)
       /* (Step 5) Identify syntactic relationships from the log */
12     Relationships ← ∅; DomainDep. ← ∅; NewVertices, NewEdges ← ∅,∅
13     doc ← NLP(log); verb ← FINDFIRSTVERB(doc)
14     direction ← None
15     foreach child in verb.children do
16         if child.dep ∈ {dobj,pobj} then
17             direction ← incoming
18         else if child.dep ∈ {nsubj,nsubjpass} then
19             direction ← outgoing
20     if direction then
21         rel ← CREATERELATIONSHIP(direction, doc, Entities)
22         Relationships ← Relationships ∪ rel
       /* (Step 6) Annotate domain-specific AR/VR dependencies */
23     foreach rel in Relationships do
24         dep ← ANNOTATEDEPENDENCY(rel, Entities)
25         DomainDep. ← DomainDep. ∪ dep
       /* (Step 7) Convert extracted data into graph vertices/edges */
26     foreach entity in Entities do
27         NewVertices ← NewVertices ∪ CREATEVERTEX(entity)
28     foreach dep in DomainDep. do
29         src ← FINDSOURCEVERTEX(dep, Vertex)
30         dst ← FINDDESTINATIONVERTICES(dep, Vertices)
31         NewEdges ← NewEdges ∪ CREATEEDGE(dep, src, dst)
32     Vertices ← Vertices ∪ NewVertices; Edges ← Edges ∪ NewEdges
       /* (Step 7 continued) Pass structured logs to feature selection */
33     PASSVERTICESANDEDGESTOFEATURESELECTION(Vertices, Edges)
34 Function ExtractEntities (FilteredTokens):
       /* (Step 3) Use custom NER labels to extract AR/VR entities */
35     Entities ← []
36     foreach token in FilteredTokens do
37         if token.label ∈ CustomNamedEntityLabels then
38             Entities.append(token)
39     return Entities
```

pounding batch size from 4 to 32. A dropout rate of 0.35 helped mitigate overfitting, while spaCy's internal optimization adjusted the learning rate dynamically. This process ensures the model can recognize key AR/VR entities (e.g., `HeadMovementData`, `VEConfigChanges`) even in logs with inconsistent formatting.

Step 4: Fuzzy String Matching. After extracting candidate entities, we employ Levenshtein distance with a 90% similarity threshold to unify multiple references to the same object. For example, *VE config* vs. *VE configurations* or *GuardianSys* vs. *GuardianSystem* become merged, preventing spurious duplications in the final provenance graph.

Step 5: Analyzing Syntactic Relations. To determine how entities interact (e.g., *who created what?*), we parse each log's syntactic dependency tree. If the log indicates *Agent-verb-Object* (e.g., `Process spawns File`), we model it as a directed edge from *Agent* to *Entity* in the provenance. Conversely, if an object is the data source (e.g., `File read by Process`), we record an incoming edge.

Step 6: Dependency Annotation. In alignment with the W3C PROV model, "`Agent`" refers to the actor performing an action, "`Activity`" denotes the action itself (e.g., `haptics override`), and "`Entity`" is the object of the action (e.g., `VE boundary`). By identifying which tokens correspond to `Agent` vs. `Entity`, our pipeline can capture the direction and semantics of each event accurately.

Step 7: Structured Data Conversion. Finally, we convert extracted entities and relationships into graph vertices and edges, passing them to the *Feature Selection* module (§5.2).

We overcome the lack of publicly available AR/VR log corpora by systematically curating unstructured logs from multiple AR/VR devices and identifying domain-specific entities (e.g., `SpatialBoundary`, `HeadMovementData`). This labeling yields a balanced dataset of 11 entity classes, with about 4–5 logs per class, providing a balanced and comprehensive coverage for accurate entity extraction. We then fine-tune a pre-trained NER model (`en_core_web_sm` [9]) on these domain-relevant examples, ensuring that REALITYCHECK can detect forensically pertinent objects under diverse AR/VR logging practices. As shown in §6.1 and §6.2, our method generalizes effectively to new attacks without further re-training.

Model Training. We employ Stochastic Gradient Descent (SGD) with cross-entropy (Eq. (1)) to optimize the NER model's weights. Leveraging *minibatch sampling* and *compounding* techniques reduces overfitting and accelerates convergence for our domain-specific corpus. This fine-tuning strategy culminates in a robust, AR/VR-oriented NER model capable of extracting key entities and relationships at scale.

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \qquad (1)$$

where N is the total number of training examples, y is the actual label, and $\hat{y}$ is the predicted label.

5.2 Feature-based Log Correlation

Creating a comprehensive provenance graph for AR/VR devices requires systematically correlating events across five diverse system layers. While the broader provenance literature [69, 109, 110] provides well-tested causality rules (e.g., parent-child processes, file activity, network connections), these rules must be adapted to the AR/VR domain, where logs are fragmented, unstructured, and often lack the typical syscall traces found on desktop or mobile systems. To address these challenges, we construct a *multi-dimensional* edge framework, capturing both *standard* host-based relationships (e.g., parent-child processes) and *AR/VR-specific* cues (e.g., device IDs or API-level events from Perfetto and OpenXR).

Log Type	Source
L1: ADB Logcat Logs	*Logcat Utility*
L2: Windows Logs	*Sysmon*
L3: Perfetto Logs	*Pftrace Utility*
L4: Device Logs	*Steam/Oculus Logs*
L5: API Logs	*OpenXR API*
L6: Application Logs	*(Instrumented) App*

Attribute	Present In
C1: timestamp	*L1 - L6*
C2: HMD_PID	*L1, L5, L6*
C3: HMD_TID	*L1, L5, L6*
C4: prev_PID	*L3*
C5: next_PID	*L3*
C6: sysmon_PID	*L2*
C7: sysmon_PPID	*L2*
C8: sysmon_event_type	*L2*
C9: process_command_line	*L2*
C10: process_details	*L1 - L6*
C11: package_name	*L1, L2*
C12: device_ID	*L1, L2, L4*
C13: device_event_type	*L4*

	Edge Features
F1	isCorrelatedThroughPID(u, v) : u.C2 = v.C2 & u.C2 = u.C3 & v.C2 = v.C3
F2	isCorrelatedThroughTID(u, v) : u.C3 = v.C3 & (u.C2 ≠ u.C3)
F3	matchPIDtoPID(u, v) : u.C4 = v.C2 & u.C5 = v.C2 & Δ(u.C1, v.C1 < t)
F4	matchPIDtoTID(u, v) : u.C4 = v.C2 & u.C5 = v.C3 & Δ(u.C1, v.C1 < t)
F5	matchTIDtoPID(u, v) : u.C4 = v.C3 & u.C5 = v.C2 & Δ(u.C1, v.C1 < t)
F6	matchTIDtoTID(u, v) : u.C4 = v.C3 & u.C5 = v.C3 & Δ(u.C1, v.C1 < t)
F7	isParent(u, v) : u.C7 ≠ 0 → u.C6 = (v.C6 == u.C7)
F8	sysmonFileCreate(u, v) : u.C6 = v.C6 & u.C8 = 11
F9	sysmonNetworkConnection(u, v) : u.C8 = 3 & u.C6 = v.C6
F10	rawAccesReadSysmon(u, v) : : u.C8 = 9 & u.C6 = v.C6
F11	similarPackageName(u, v) : u.C10 ∈ v.C9 & Δ (u.C1, v.C1 < t)
F12	sysmonCommandMatch(u, v) : u.C9 ∈ v.C10 & Δ (u.C1, v.C1 < t)
F13	matchSysmonToLogcat(u, v) : u.C8 = v.C9 & Δ(u.C1, v.C1 < t) & u.C12 = v.C12
F14	joinOculusLogs(u, v) : u.C12 ∈ v.C10 & u.C13 ∈ v.C10 Δ (u.C1, v.C1 < t)
F15	matchDataTransferIndicator(u, v) : u.C10 ∈ v.C10 & u.C12 = v.C12 & Δ(u.C1, v.C1 < t)

Figure 5: Attributes and features for determining causal relationships between different logs.

Our approach uses a 15-dimensional feature vector $\vec{v} = [F_1, \ldots, F_{15}]$, shown in Figure 5, where each dimension F_k encodes a specific relationship. The presence of a k-th type connection is denoted by R_k, with $R_k = 1$ if valid and $R_k = 0$ otherwise. We began by reviewing how prior provenance systems [69, 109, 110] track events and data flows across distinct layers (e.g., OS kernel vs. user space). Building on those insights, we designed new correlation features suitable for AR/VR constraints:

- F_1 and F_2 model the correlation between log entries from HMD (Logcat, OpenXR, and Application) logs that share the same PID or TID. These features capture explicit correlations between the creation and operation of processes.
- F_3 to F_6 identify associations between Perfetto trace logs and HMD logs. By comparing the previous and subsequent PIDs or TIDs in Perfetto logs with PIDs or TIDs in other logs, these features facilitate the establishment of connections between HMD logs.
- F_7 to F_{10} represent correlations in Sysmon records pertaining to parent-child process relationships, file creation, network connections, and file manipulation events.
- F_{11} to F_{15} connect Windows Event logs and HMD logs by satisfying multiple dimensions. These capabilities enable a more comprehensive view of events involving multiple log sources, and are further explained below.

F_1-F_{10} capture straightforward relationships such as shared PID/TID among HMD logs (Logcat/OpenXR/App logs), Perfetto-based "prev/next" transitions, and Sysmon's parent-child processes or file/network events. They align with conventional provenance logic but **must be ported** to AR/VR, where root privileges and system-call tracing are unavailable. Because AR/VR logs often omit unique process IDs or rely on ephemeral device IDs, we added *compound correlations* described below comprising $F_1$1-F_{15} (e.g., package name + device ID + timestamp) to associate events across different logs. These features detect subtle connections—like matching command lines in Sysmon with partial tokens in AR/VR logs or linking data transfers across Windows Event logs and Oculus logs. Without these multi-faceted rules, many essential cross-layer edges would remain undiscovered.

- F_{11} and F_{13}: Match package name, event types, and device IDs between Sysmon and HMD logs.
- F_{11} and F_{14}: Connect Logcat and Oculus logs by matching package name, device IDs, and Oculus event types.
- F_{12} and F_{15}: Match command lines in Sysmon and HMD logs; link HMD and Oculus logs via data transfer indicators and device IDs.
- F_{13} and F_{14}: Match the event types and device IDs between Sysmon and HMD logs, as well as between HMD logs and Oculus logs.

5.3 Multi-Layer Provenance Graph (MPG)

Utilizing NLP techniques (§5.1), REALITYCHECK first processes unstructured logs to extract key entities and their relationships. These extracted details, combined with structured logs, are then subjected to our feature selection technique (§5.2). This comprehensive approach enables REALITYCHECK to construct a Multi-layer Provenance Graph (MPG) that represents the intricate relationships and sequences of an attack. The MPG provides a holistic visualization, mapping out the entities (vertices) and their interdependencies (edges) in the context of the attack. However, the MPG is not yet free of the dependency explosion problem and contains irrelevant entities, which we address in §5.4 and §5.5, respectively.

5.4 AR/VR-Aware Execution Partitioning

Traditional auditing systems often assume that each output event depends on *all* preceding input events [69, 76, 80, 87, 90, 105], leading to a *dependence explosion* [79, 80, 85, 86]. This is particularly problematic for AR/VR applications that run continuously and generate high-frequency logs (e.g., sensor updates). To address this, prior work introduced *execution partitioning* on event-handling loops [80, 84]. This technique identifies event-handling loops in Windows and Linux applications and creates *execution units* for each loop iteration. Each unit only contains causally related events, reducing false dependencies. However, the unique nature of the AR/VR programming paradigm involving OpenXR session states requires us to take a different approach. Unlike typical desktop

```
static XrSessionState g_sessionState = XR_SESSION_STATE_UNKNOWN;
static std::thread::id g_inputThreadID, g_renderThreadID;
void OnSessionStateChanged(XrSessionState newState) {
  if (newState == XR_SESSION_STATE_RUNNING) return;
  g_sessionState = newState;
  std::cout << "[Partition] sessionState-> " << int(g_sessionState)
            << " (new Execution Unit via TIDs)\n";}
void InputThreadLoop() {
  g_inputThreadID = std::this_thread::get_id();
  while(true){std::this_thread::sleep_for(std::chrono::seconds(3));
    if (g_sessionState == XR_SESSION_STATE_FOCUSED) {
      std::cout << "[Input] TID=" << g_inputThreadID
                << " -> user action in FOCUSED.\n";}}}
void RenderThreadLoop() {
  g_renderThreadID = std::this_thread::get_id();
  while(true){std::this_thread::sleep_for(std::chrono::ms(500));
    if (g_sessionState == XR_SESSION_STATE_VISIBLE
        || g_sessionState == XR_SESSION_STATE_FOCUSED) {
      std::cout << "[Render] TID=" << g_renderThreadID
                << " -> drawing scene.\n";}}}
```

Listing 1: Condensed excerpt of an OpenXR app from Khronos [32]. We create "execution units" on transitions (READY, FOCUSED, STOPPING), skipping per-frame RUNNING. Two threads (input vs. rendering) remain distinct via TIDs.

software, OpenXR haandles application flow through *session states* (e.g., `READY`, `VISIBLE`, `FOCUSED`, `STOPPING`), not discrete handleEvent() calls. These session transitions reflect meaningful changes in user engagement—when the user actually *enters* or *exits* immersive rendering—rather than the ephemeral, high-frequency iterations seen in normal event loops. Hence, partitioning exclusively on conventional "per-event" or "per-iteration" loops fails to capture the unique, high-frequency concurrency of AR/VR systems. Instead, as we demonstrate, leveraging *OpenXR session-state transitions* at higher-level boundaries (e.g., `READY`→`FOCUSED`) provides a more coherent partitioning scheme that reduces false dependencies while capturing the application's truly distinct operational phases.

Skipping Single-Invocation and Per-Frame States. In typical OpenXR development, calls like `xrCreateInstance` or `xrCreateSession` occur only once during an application execution lifecycle, offering coarse partitions that blur many unrelated events together. Conversely, `XR_SESSION_STATE_RUNNING` *fires on every frame*, creating thousands of micro-partitions. Hence, we concentrate on *session transitions* such as `XR_SESSION_STATE_READY`, `XR_SESSION_STATE_VISIBLE`, `XR_SESSION_STATE_FOCUSED`, and `XR_SESSION_STATE_STOPPING`. Each transition naturally encapsulates a coherent phase of user activity or system state change, eliminating extraneous dependencies.

Listing 1 shows a simplified excerpt from a real "Color Cube" OpenXR application in the official Khronos development guide [32]. This app lets the user *spawn*, *recolor*, and *move* cubes by pressing controller buttons. It cycles through session states from `READY` to `FOCUSED` to `STOPPING`, skipping `RUNNING` frame-by-frame loops to avoid overly fine partitions. We highlight two threads: an "input" thread (polling or randomizing events) and a "render" thread. Instead of tracking every `xrWaitFrame`/`xrEndFrame` iteration, we *only* partition logs on transitions like `READY`→`VISIBLE`, `VISIBLE`→`FOCUSED`, and `FOCUSED`→`STOPPING`, each of which signals a logical jump in user engagement. This avoids an explosion of micro-partitions while still isolating distinct phases. Moreover, multiple threads remain separated by their thread IDs (TIDs), preventing concurrency from merging unrelated I/O events into a single execution unit. This way, each *execution unit* corresponds to a meaningful slice of user interaction.

Execution Model Analysis for App State Identification. We further distinguish *foreground* activities (e.g., the user actively manipulating cubes while `FOCUSED`) from *background* or idle states, restricting logs that are relevant to an attack scenario. For example, a malicious "re-bind" of the color-change action (leading to data exfiltration) would appear only during `FOCUSED`, not lost among the thousands of `RUNNING` frames.

Cross-partition Attack. By focusing on meaningful *session-state* transitions (e.g., `READY` →`FOCUSED`) instead of per-frame boundaries, REALITYCHECK mitigates false dependencies without fragmenting the attack storyline. As illustrated in §3 and Figure 2(b)–(c), our partitioning preserves essential causal edges while discarding spurious connections (shown as black arrows) in (b), ensuring that graph segments remain inherently connected yet free of dependency explosions.

5.4.1 Non-OpenXR Application Instrumentation

OpenXR API [16], as discussed in §5, is adopted by most AR/VR applications and manufacturers. However, AR/VR devices also support non-OpenXR API applications, often sourced from third-party stores like SideQuest [42]. These applications' logs typically lack detailed function invocations and their arguments, as seen in OpenXR API logs.[3] To address this challenge, we use an instrumentation-based approach, leveraging Soot, to enhance the granularity of non-OpenXR app logs akin to OpenXR ones. Soot can instrument an Android app's Java bytecode, making it suitable for both open- and closed-source apps, provided the SDK is identifiable through the app's manifest or bytecode.

The challenge then becomes pinpointing which functions in non-OpenXR apps to instrument. Adapting the lifecycle-based partitioning from §5.4, we selectively instrument lifecycle-related functions. For native Android SDK apps, we focus on functions analogous to OpenXR's `xrPollEvent`, like Android's `onResume` and `onPause`. For Unity and Unreal SDK apps, we target functions linked to game engine events, as detailed in Appendix D. This mirrors the OpenXR API's partitioning approach.

Our approach processes Android application files by targeting specific functions. For each method identified in the bytecode, we extract its method body, referred to as the unit chain U. We then create a new unit chain U', which includes augmented units. As we traverse each unit in U, we check for invocations of the target methods. When such invocations are detected, we insert logging statements into U'. This new unit

[3] Our study indicates that eighteen of the top thirty AR/VR apps on SideQuest [42] utilize native or mixed Android SDK.

chain, consisting of both original and augmented units, replaces the original to preserve the application's functionality. After modifying all relevant methods, we validate the altered method bodies to ensure the integrity of the instrumented app file. This process allows for precise bytecode modification, focusing solely on key functions to facilitate lifecycle-based partitioning in non-OpenXR SDKs. The pesudocode for our algorithm is provided in Appendix D. We also discuss the use of native library instrumentation in §6 and Appendix A.

5.5 MPG Pruning

REALITYCHECK uses an automated graph pruning (GP) mechanism to remove forensically irrelevant vertices and edges from AR/VR multi-layer provenance graphs. Our approach, based on the Direct Forensic Connection (DFC) and Indirect Forensic Connection (IFC) principles, retains crucial evidence while discarding extraneous details.

DFC. We first mark all components directly linked to the detection point as *critical*. The detection point is the abnormal vertex flagged by the intrusion detection system. Any element bearing an immediate connection to the investigation's focal vertex remains exempt from pruning.

IFC. Next, we analyze remaining graph elements and keep those that: *(i)* act as intermediaries for data flow between critical vertices, or *(ii)* link two critical vertices. These indirect connections are needed for reconstructing event causality.

Following the principles of DFC and IFC, the GP process applies the rules described below to prune the graph. ***Rule 1 (Regular Frequency Vertices):*** AR/VR devices log routine metrics at fixed intervals (e.g., battery usage every 15s). Such logs do not impact user interactions or attacks. ***Rule 2 (Standalone Operational Vertices):*** Vertices denoting system maintenance tasks lack outgoing edges and do not influence user or attack processes. ***Rule 3 (Irrelevant Process/Object Vertices):*** Inspired by RapSheet [68], a vertex *P* or *O* is pruned if *(i)* its backward-tracing graph has no link to the detection point and *(ii)* no event edges with *P* or *O* involve the detection point. ***Rule 4 (Repetitive Entities):*** AR/VR logs often repeat processes (e.g., repeated `DownloadManager` entries). Only the final completion entry matters (e.g., download completed); hence, we merge repeated operations into a single vertex.

6 Evaluation

We rigorously evaluated our REALITYCHECK prototype across six critical metrics: (1) the effectiveness of attack reconstruction, (2) the accuracy and efficiency of the NLP approach, (3) runtime, storage, and query response overheads, and (4) generalizability across a wide range of popular AR/VR devices and adaptability to future threats.

Generalizability. In terms of generalizability to emerging threats, we utilized our system without modification for recent attacks that emerged after 2023 and show that it was able to provide accurate MPGs. As shown in Table 3, the investigation effectiveness for attacks in 2024 is comparable to those prior to 2023. Finally, in terms of generalizability to other hybrid AR/VR devices, we performed our MPG effectiveness analysis on HTC Vive XR Elite [47], which revelead that REALITYCHECK can generalize to other AR/VR devices besides Meta Quest 2, perform near-perfect investigation. These results are discussed in detail in Appendix B.

Selected Device. We developed our system prototype on the Meta Quest 2, which holds over 90% of the AR/VR market share [15]. This prevalent device was chosen for its representative architecture. Our evaluation was conducted on the Meta Quest 2 with Android OS build 333700.3370.0, using a machine with an Intel Core i5-10400F Hexa-Core Processor (2.9 GHz), 16.0 GB RAM, and running Windows 11.

Implementation. We built a REALITYCHECK prototype in Python 3.9.12 and Java 1.8.0u341 (for instrumentation via Soot), totaling 7K lines of code. During runtime, automated scripts use the Android Debug Bridge to gather device and PC logs. Once an attack occurs, the evaluation phase processes these logs into an MPG using NetworkX [23] and PyVis [38] for visualization. We use spaCy's `en_core_web_sm` model [9] for custom NER and syntactic relationships, and NLTK [25] for POS tagging and lemmatization. Our NER model was trained on annotated AR/VR logs split into 70% training, 10% validation, and 20% testing, running 200 iterations with a dynamically increasing batch size. The learning rate was managed by spaCy's optimization, while our POS tagging is unsupervised.

Native-library Instrumentation. While our initial plan was to instrument native (C/C++) libraries by modifying the Android system at the root level, the closed Android-based OS on AR/VR devices limited root privileges and made conventional system-wide DLL instrumentation unworkable [26, 102]. Consequently, we employed a user-space hooking method with the Frida toolkit, dynamically intercepting and logging function calls in each target DLL. In particular, we introduced lightweight hooks for lifecycle and I/O-related functions (e.g., file/network handling, sensor operations, and configuration updates). We used open-source AR/VR apps from our dataset (apps #9–15) to identify function names and signatures for accurate Frida script injection. At runtime, Frida loaded these hooking scripts into the app's process without altering the firmware or requiring root access, thereby capturing syscall-like data from native libraries under a root-free setup. However, as shown in Appendix A, this approach imposes substantial overhead without improving investigation efficacy, a trade-off we discuss further in that section.

Datasets & App Choice Criteria. Our evaluation uses two types of datasets: *attack datasets* and *application benchmarks*. The attack datasets were curated to test the effectiveness of our methodology for investigating AR/VR-specific

attacks. Due to the lack of open-source AR/VR attack datasets, we generated these by simulating a variety of attacks using real AR/VR apps (Table 2), following the procedures in Appendix E and Table 9. We employed APT scenarios from MITRE Caldera [8], adversary emulation tools [13, 17, 37], and GitHub exploit repositories [1, 21], and replicated attack techniques detailed in prior research [55, 71, 98, 103]. This approach generated realistic, complex attack scenarios, enabling a comprehensive assessment of REALITYCHECK's investigative capabilities. In contrast, the application benchmarks were selected to evaluate REALITYCHECK's performance and logging overhead. We systematically chose 15 applications from the Meta Quest Store and SideQuest, prioritizing those that employ OpenXR APIs or come pre-installed on the device. To showcase the versatility of our Soot-based instrumentation with non-OpenXR apps (given that the Meta Quest Store mostly offers OpenXR-only applications), we also included top downloaded opens-source apps from the SideQuest store.

Comparison with Existing Systems. Although some attacks (e.g., data exfiltration) also occur on conventional Android devices, existing Android-centric auditing tools cannot be used on AR/VR headsets due to *device-level restrictions*. Most AR/VR platforms do not allow the root privileges or syscall tracing that systems like Dagger [108], DroidForensics [111], or AppAudit [106] rely on, as detailed in §1. In addition, commercial AR/VR devices lack the open firmware environments needed to install kernel hooks or run system-call loggers. Hence, even attacks that appear on both Android phones and AR/VR headsets remain beyond the reach of these tools because the latter cannot run at all (or cannot access low-level logs) on closed AR/VR systems. OVRSeen [102] is the only feasible option under these constraints and was therefore chosen as a baseline for our evaluations in Table 6.1. However, its reliance on complex dynamic analysis limits its practicality on end-user AR/VR devices. Finally, the proprietary nature of many AR/VR apps further restricts solutions like AppAudit [106], which requires source code access for instrumentation. In contrast, our approach is *device-aware*, sidestepping the need for root access and capturing multi-layer logs without modifying the closed firmware or kernel.

Ground Truth MPGs. Given the lack of pre-existing ground truth provenance graphs for AR/VR attacks, we manually generated these graphs based on detailed ground truth documents that accompanied the attacks, which thoroughly described the attack behaviors. Our manual approach aligns with established practices in attack provenance graph labelling [67, 73, 74, 93, 105]. To further validate the correctness of our ground truth graphs, two independent authors carefully reviewed the manually generated provenance graphs. They assessed whether these graphs accurately represented the root causes and impacts of each AR/VR attack scenario. The high inter-rater reliability achieved in this review process confirms the robustness and correctness of our approach.

Table 3: Effectiveness of OVRSeen vs REALITYCHECK in tracing the provenance of different attack scenarios. G.T.: Ground Truth; RC: REALITYCHECK; EP: Execution Partitioning; GP : Graph Pruning; V: Vertices; E: Edges; Cov.: Coverage; TP: True Positives; FP: False Positives; FN: False Negatives; Prec.: Precision; Rec.: Recall. Each row also includes V/E numbers from the APT portion of the attack.

	Investigation Effectiveness						NLP Effectiveness			
	GT	OVRSeen		RC	RC + EP & GP		TP/FP/FN	Prec.	Rec.	F_1
	V/E	V/E	Cov.	V/E	V/E	Cov.				
1	98/109	-	-	152/162	98/108	1.00	36/2/1	0.95	0.97	0.96
2	90/101	-	-	125/139	90/100	0.99	21/1/0	0.95	1.00	0.98
3*	112/127	1/0	0.00	194/252	112/127	1.00	69/3/0	0.96	1.00	0.98
4*	111/126	1/0	0.00	188/224	111/126	1.00	67/2/0	0.97	1.00	0.99
5	107/118	1/0	0.00	183/197	107/118	1.00	55/0/0	1.00	1.00	1.00
6	106/117	1/0	0.00	177/188	106/117	1.00	53/0/0	1.00	1.00	1.00
7	98/109	1/0	0.01	147/161	97/109	1.00	36/2/1	0.95	0.97	0.96
8	102/113	1/0	0.01	157/172	102/113	1.00	45/3/0	0.94	1.00	0.97
9	98/109	1/0	0.01	149/164	98/109	1.00	37/2/0	0.95	1.00	0.97
10	98/109	1/0	0.01	142/155	98/109	1.00	37/1/0	0.97	1.00	0.99
11	104/115	1/0	0.00	177/191	103/115	1.00	48/3/1	0.94	0.98	0.96
12	93/104	2/0	0.01	133/145	93/104	1.00	27/1/0	0.96	1.00	0.98
13	92/102	1/0	0.01	136/147	91/102	0.99	23/2/1	0.92	0.96	0.94
14	101/112	2/0	0.01	143/159	101/112	1.00	43/2/0	0.96	1.00	0.98
15	101/113	2/0	0.01	136/151	101/113	1.00	44/3/0	0.94	1.00	0.97
16	98/109	-	-	148/159	97/108	0.99	35/1/2	0.97	0.95	0.96
17	91/102	1/0	0.01	138/152	91/101	0.99	22/1/1	0.96	0.96	0.96
18	98/109	-	-	144/159	97/109	1.00	36/1/0	0.97	1.00	0.99
19*	96/106	1/0	0.00	133/145	96/106	1.00	32/1/0	0.97	1.00	0.98
20	98/109	2/0	0.01	137/148	97/108	0.99	35/1/2	0.97	0.95	0.96
21	93/103	2/0	0.01	128/143	92/103	0.99	25/1/1	0.96	0.96	0.96
22	95/105	1/0	0.01	141/153	95/105	1.00	30/2/0	0.94	1.00	0.97
23	91/102	1/0	0.01	123/134	91/102	1.00	23/0/0	1.00	1.00	1.00
24	104/115	1/0	0.00	151/163	103/115	1.00	48/2/1	0.96	0.98	0.97
25	92/102	1/0	0.01	136/148	92/102	1.00	24/0/0	1.00	1.00	1.00

Note: Rows marked with * indicate attacks that emerged after 2023. The results show generalizability of REALITYCHECK when adapting to future attacks since our labelling was performed before these attacks emerged.

6.1 Investigation Effectiveness

To evaluate the effectiveness of REALITYCHECK against OVRSeen[4] in tracing attack provenance, we used real *advanced persistent threat (APT)* scenarios. Specifically, we used MITRE Caldera [8] to simulate a realistic multi-stage APT, incorporating the following steps: (1) **Initial Access, Privilege Escalation, and Presistence:** compromising a Windows PC connected to the headset to establish foothold, (2) **Device Discovery:** identifying the tethered AR/VR device from the PC to facilitate lateral movement, (3) **Lateral Movement Execution:** deploying custom payloads via `mal.exe` to install/modify AR/VR apps on the headset, (4) **System Manipulation:** optionally altering device configurations to maintain access or prepare for cognitive attacks, (5) **Data Collection:** gathering sensitive AR/VR data such as user interactions and logs, and (6) **Data Exfiltration:** utilizing the compromised PC and the headset to exfiltrate information.

We launch the 25 AR/VR attacks (Table 2) on the compromised device using real AR/VR apps provided in the corresponding table, mirroring the multi-step intrusions seen in the wild. Because the APT steps add additional vertices and

[4]Although most attacks can be run across multiple SDKs, dashed entries for OVRSeen represent attacks untestable on the OVR SDK due to API deprecation

Table 4: Ablation study showing NLP's contribution to investigation efficacy. RC: REALITYCHECK. RC without ER, POS uses edge dictionary method like existing works [62, 95] for edge extraction.

Configuration	Precision	Recall	F1	Graph Size Red. (%)
RC w/o ER, POS	0.79	0.64	0.71	57.1
RC + ER only	0.79	0.64	0.71	70.8
RC + POS only	0.96	0.99	0.98	57.1
RC (all)	0.96	0.99	0.98	70.8

edges in MPGs for each attack beyond just the AR/VR device footprint, our ground truth graphs expand accordingly. These attacks were analyzed using REALITYCHECK to generate MPGs, summarized in Table 3. Coverage is calculated as $\frac{V_c+E_c}{V_g+E_g}$, where V_c and E_c are the correctly identified vertices and edges, respectively, and V_g and E_g are the vertices and edges in the Ground Truth. REALITYCHECK consistently achieved near-perfect coverage, indicating successful capture of all vertices and edges related to the attack. When coverage fell to 0.99, it was typically due to REALITYCHECK's conservative use of double-sided edges in response to ambiguous verbs like "accessing," whereas Ground Truths specified stricter edge directions. This conservative approach, aligned with literature on edge inference from texts [62, 81, 95], ensures no potential interactions are missed.

As established in [69, 80], attack provenance graphs must demonstrate *Soundness* and *Completeness* for reliable forensic analysis. Soundness ensures the graph respects the "happens-before" relationship for accurate querying, while Completeness ensures all necessary details for determining root causes and attack ramifications are included. Our REALITYCHECK-generated MPGs for all attacks were both sound and complete, accurately reflecting happens-before relations and root causes when compared to the ground truth[5].

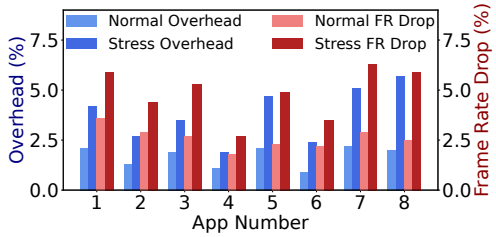
6.2 Effectiveness of the NLP Approach

We evaluated the NLP approach of REALITYCHECK, which extracts entities and relationships from unstructured logs, using precision, recall, and F_1 score. The datasets were split into testing and training sets for validation. Metrics like true positives, false positives, and false negatives gauged the system's representation accuracy of the original log activities. Table 3 showcases high scores across attack scenarios, underscoring the NLP approach's efficacy. Given the challenge of quantifying true negatives, accuracy is not a primary metric. Instead, we emphasize precision, recall, and F_1 score, aligning with other studies [62, 95]. A few false positives and false negatives stem from our method of creating bi-directional edges when logs contain ambiguous verbs, as discussed in §6.1 and not missed vertices or edges in MPGs. Hence, these minor discrepancies are considered trivial as they do not impact the overall validity, soundless and completeness of the generated MPGs.

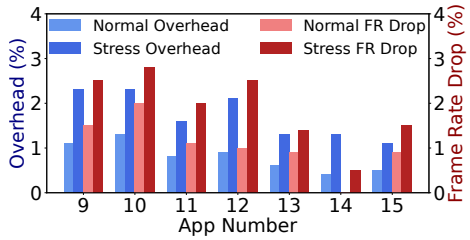
[5]Soundness is ensured by tracing PID/TID and synchronizing log timestamps. Completeness is achieved by integrating logs from all system layers using our log correlation techniques detailed in §5. Even when coverage is not 1.0, Completeness is preserved through conservative use of double-sided edges to retain all relevant data.

Table 5: Logging overhead using REALITYCHECK. We removed the prefix "com." from each package name for readability. EP: Execution Partitioning; GP: Graph Pruning.

App Number	Package Name	Log Size (Kb)	# Logcat / API / Perfetto logs	Total Logs	# Logs after EP and GP
1	oculus.browser	18321	2871 / 62 / 8456	11389	3419
2	beatgames.beatsaber	18172	2738 / 55 / 8432	11225	3031
3	facebook.horizon	13454	2863 / 58 / 7964	10885	2983
4	oculus.explore	13102	3243 / 71 / 8327	11641	2797
5	oculus.vrshell.home	12875	2987 / 68 / 7551	10606	2831
6	oculus.systemutilities	12240	2801 / 47 / 7762	10610	2630
7	transcendxr.reality	16331	3235 / 71 / 9242	12548	3663
8	bigscreenvr.bigscreen	18100	2732 / 93 / 8298	11123	3078
9	qcxr.qcxr	20313	3612 / 79 / 9827	13518	4271
10	arda.computer.arda	21240	3774 / 82 / 10356	14212	4547
11	homeassistant. companion.android	9482	2007 / 36 / 5872	7915	2380
12	xrstream.ovr	12446	3090 / 67 / 7634	10791	3173
13	activitywatch.android	7410	1380 / 21 / 5490	6891	2342
14	questapp versionswitcher	7552	1334 / 23 / 5674	7031	2420
15	justmedafaq. questnotifier	3568	637 / 14 / 2917	3568	1377



(a) Runtime memory overhead



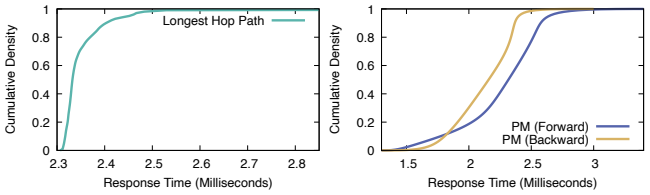
(b) Instrumentation overhead

Figure 6: Runtime overheads associated with REALITYCHECK.

Ablation Study. Table 4 shows how varying different components of our NLP pipeline affects the NLP efficacy of our approach. When both ER and POS are omitted, precision and recall drop to 0.79 and 0.64 (F_1=0.71), while adding only ER retains the same F_1 but increases graph size reduction from 57.1% to 70.8%. Using only POS yields precision 0.96 and recall 0.99 (F_1=0.98), though graph reduction reverts to 57.1%. Combining ER and POS achieves F_1=0.98 and the highest graph size reduction (70.8%), indicating both components are vital for extracting relationships in unstructured AR/VR logs.

6.3 Performance Evaluation

Storage Overhead. We used Monkey [19], integrated into Meta's Android framework, for stress testing and to evaluate



(a) Longest-hop turnaround time. (b) Forward and Backward query time.

Figure 7: Query performance of REALITYCHECK.

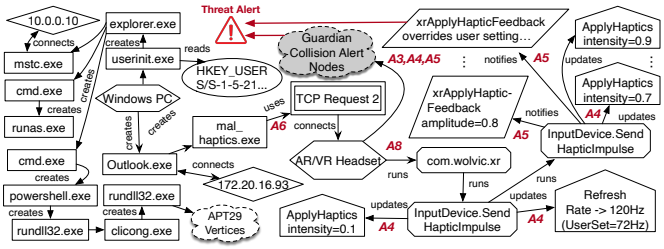


Figure 8: A concise form of the MPG generated by REALITYCHECK for the APT29 attack engagement [6] + Photosensitive epilepsy [55].

logging overheads. Table 5 shows the logging overhead for various applications tested with Monkey over 10,000 events. This table includes log sizes, entries from Logcat, API, Perfetto logs, average logs per event, and logs post our system's pruning and partitioning. Different applications exhibit varied logging overheads due to their unique resource requirements, but our system consistently filters out irrelevant entries. Logs were reduced by 61.39% to 76.00% across apps, with an average reduction of 70.79%. In Appendix C, we also provide the storage growth of REALITYCHECK over time, which we argue is minimal, as the storage growth per 24 hours is just 12MB when logs are compressed.

Runtime Overheads. We employed the OVR Metrics Tool [35] under both normal and stress-test workloads. Figure 6a shows overheads ranging from 0.4% to 2.2% (averaging 1.3%) in normal operation and 1.1% to 5.7% (2.9% on average) under stress tests. Correspondingly, frame-rate (FR) drops vary between 0.0% and 3.6% (avg. 1.8%) for normal usage, and rise as high as 6.3% (avg. 3.3%) in stress testing. Figure 6b further indicates that our selective instrumentation remains below 2.5% overhead, with the highest observed overhead at 5.7%. Prior work [53, 94] suggests this level of overhead is acceptable for traditional Android-based systems, though we acknowledge the absence of a user study and leave that as a future direction.

Graph Query Response Times. To comprehensively analyze the consequences and root causes of all 25 documented attacks, we performed both forward and backward provenance queries, alongside assessing the longest hop paths within our system. The query response times, presented in Figure 7a, show REALITYCHECK's efficiency, with all vertices queried in 2.83 ms, enabling rapid graph generation for threat investigation. Response times for forward and backward provenance queries were smaller, averaging in milliseconds, with maxima of 2.57 ms and 2.37 ms respectively (Figure 7b). The improved efficiency, relative to the 2.83 ms for all vertices, is because the detection points are at varying graph positions, not just at the end of the longest hop paths.

7 Attack Case Study: Photosensitive Epilepsy

Scenario. An APT29 actor [6] first compromises a Windows PC by leveraging a phishing email within `Outlook.exe`, later pivoting through `cmd.exe`, `runas.exe`, and `powershell.exe` to escalate privileges (see Figure 8). Ultimately, the attacker launches `mal_haptics.exe`, which leverages Android debugging utilities to sideload a malicious AR/VR app `org.mozilla.vrbrowser`, which is actually the *Firefox* browser with injected malicious code [104]. This app stealthily overrides user preferences for disabled haptics, repeatedly invoking calls like `xrApplyHapticFeedback` at higher amplitudes (e.g., intensity=0.1, 0.7 0.9) despite the user's "OFF" setting (intensity=0.0). Simultaneously, it changes refresh rates from 72 Hz to 120 Hz and introduces fast strobe flickers, risking photosensitive seizures and severe discomfort. By conflating haptic surges with rapid frame updates, the attacker effectively launches a cognitive attack, rendering the user disoriented and at physical risk.

Investigation. When suspicious boundary collisions trigger an anomaly, REALITYCHECK correlates the Windows-based APT portion with repeated calls to `xrApplyHapticFeedback` that override the user's preference. The provenance graph (Figure 8) links `mal_haptics.exe` to suspicious AR/VR session calls at amplitude ≥ 0.8, as well as *"Refresh Rate $\rightarrow$ 120Hz (UserSet = 72Hz)"* updates, indicating a sudden spike in display flicker beyond the user's default setting. Additionally, REALITYCHECK detects `org.mozilla.vrbrowser` invoking `xrApplyHapticFeedback` at times that coincide with higher haptic amplitudes, even though the user had disabled haptics due to sensitivity. Once the logs show a strobe frequency above normal and forced haptic amplitude, resulting in collisions with Guardian boundaries, the underlying threat detection system raises an alert, labeling these events as abnormal. Through these cross-layer relationships, REALITYCHECK reveals how a cognitively targeted attack, orchestrated by APT29, manipulates the AR/VR environment to induce severe sensory overload—ultimately triggering a *Threat Alert* due to abnormal haptic usage and forced display configuration in a real APT-style AR/VR attack scenario.

8 Related work

In §1 and §3 we described the challenges with existing provenance trackers for mobile systems that REALITYCHECK addresses, and complement the discussion on related work here.

Provenance Graph Applications & Graph Pruning. Prove-

nance has been applied to network troubleshooting [54] auditing [66], and IoT forensics [105]. Tools like Hercule [92], OmegaLog [69], and ALchemist [110] correlate logs on endpoints (e.g., Windows hosts). By contrast, REALITYCHECK integrates logs across the AR/VR stack for accurate MPGs. Existing systems [70, 72, 74, 107] reduce endpoint graph size but do not address AR/VR complexities discussed in §5.5. REALITYCHECK employs pruning tailored to AR/VR contexts, retaining crucial attack details. Future work may integrate these prior methods to further refine log reduction for AR/VR investigations.

Provenance Tracing in Android. TaintDroid [61] tracks sensitive data leaks. In contrast, VetDroid [112] checks app permission use. Dagger [108] uses provenance for Android app vetting. We describe the limitation of these existing systems for AR/VR auditing in §3.

AR/VR Attack Categorization. Kulal et al. [77], Giaretta et al. [64], and De Luca et al. [58] survey AR/VR attacks but lack the structure of MITRE ATT&CK framework [18]. They also omit new tactics from Luo et al. [83], Slocum et al. [99], and Cheng et al. [56, 57]. Our MITRE-based classification covers these techniques comprehensively.

NLP for Unstructured Log Analysis. CTI-based NLP approaches [62, 81, 95] differ from AR/VR logs, which have domain-specific terms and lack syscall references. REALITYCHECK handles logs like *"HeadsetCollisionTracker: HMD Collision Notification"* which do not fit neatly into the finite vocabularies of syscall dictionaries [62, 95]. Unlike prior systems [59, 60, 65, 114], REALITYCHECK tailors NLP to AR/VR logs for precise provenance generation.

9 Discussion and Limitations

In this section, we discuss the limitations and possible extensions of our system through a series of questions.

What are the limitations of REALITYCHECK's static instrumentation? Static instrumentation in REALITYCHECK is lightweight, capturing key AR/VR lifecycle transitions, but it can miss dynamically loaded code. Malicious logic may also bypass static hooks. Nevertheless, monitoring only critical lifecycle points keeps overhead low and our of–the-shelf-logs still collect syscall-like data (e.g., file/socket events) reflecting potential dynamic payloads. We also tested native-library instrumentation (via Frida), which raised overhead to 45.49% under stress tests yet yielded no added forensic insights beyond the static-level hooks and multi-layer logs.

What are the usability impacts of instrumentation is REALITYCHECK? We evaluated runtime and frame-rate (FR) overhead but did not conduct a formal user study. Under normal conditions, FR drop remains below 3.6%, peaking around 6% in stress tests. Since many AR/VR apps run above 120 Hz, even a 6% decrease keeps them well above the 72 Hz threshold for Meta Quest store app acceptance [41]; some works suggest rates of 90 Hz or more help avoid motion sickness, and our overhead measurements confirm that the tested apps maintain frame rates above this threshold. Still, performance-sensitive scenarios could be affected, motivating future studies on user-perceived latency.

How is REALITYCHECK able to investigate cognitive attacks? REALITYCHECK models immersive attributes (e.g., boundary states, sensory data) to detect manipulations that endanger user perception or well-being. By correlating logs on boundary adjustments, haptic feedback, and visual configuration changes, REALITYCHECK identifies scenarios that might physically or psychologically harm users – for instance, shifting guardian walls or bombarding them with high-amplitude haptics. These disruptions often appear as sudden boundary collisions or flickering refresh-rate toggles, which REALITYCHECK logs and adds to its provenance graphs.

Does REALITYCHECK perform attack detection? REALITYCHECK is designed for *investigation*, not direct threat detection. Following the guidance of standards like NIST SP800-61r2 [24] on the importance of root cause analysis, REALITYCHECK operates once an existing alert (threat symptom) arises. Once a threat is flagged by an underlying system [60, 96, 114], REALITYCHECK can reconstruct and analyze the attack path, much like other forensic-focused research [67, 69, 74, 75, 82, 86, 92, 105].

How does REALITYCHECK address privacy concerns in data provenance systems? Data provenance systems, such as REALITYCHECK, handle sensitive user information in logs, where privacy is a critical concern. Securing log storage to protect user privacy, while crucial, is orthogonal to our core work. Integrating privacy-preserving mechanisms in our system is an area we plan to explore in future work.

Who are the users of the REALITYCHECK system? REALITYCHECK' users include security analysts, digital forensics experts, AR/VR developers, IT administrators, and enterprise security vendors. They leverage REALITYCHECK for root cause analysis and impact assessments, generating detailed provenance graphs of AR/VR attacks that inform enhanced threat prevention, refined detection signatures, and stronger AR/VR security frameworks.

10 Conclusion

We present REALITYCHECK, an end-to-end auditing system designed for effective investigation of AR/VR attacks using provenance graphs. REALITYCHECK introduces novel NLP techniques, a feature-based log correlation approach, AR/VR-aware execution partitioning strategies, and graph pruning methodologies to generate accurate provenance graphs for AR/VR attacks. We developed a prototype for Meta Quest 2 and evaluated it against various AR/VR attacks. The results demonstrate that REALITYCHECK is generalizable, non-intrusive, and facilitates precise attack investigations.

11 Acknowledgment

We sincerely thank the anonymous shepherd and reviewers for their insightful feedback on this work. We also thank Wentao Chen, Santino Martella, and Sherry Zhao for their assistance with attack simulations and log collection. Our research was supported by NSF CNS grant 23-39483 and the Commonwealth Cyber Initiative (CCI) award.

12 Ethics Considerations and Open Science

Ethics. In developing and testing REALITYCHECK, we strictly followed ethical guidelines: no human subjects were involved, no new vulnerabilities were discovered, and no live testing occurred on public networks. All attacks studied were known, well-documented, and performed in secure, controlled environments for research purposes only. To ensure transparency, we cited all relevant attacks, toolkits, and repositories and provided comprehensive details in Appendix E.

Open Science. Our code and log collection scripts are available at https://doi.org/10.5281/zenodo.14736506.

References

[1] Oculus Quest Research. https://github.com/QuestEscape/research.

[2] ActivityWatch. https://sidequestvr.com/app/201/activitywatch.

[3] Git repositories on android. https://android.googlesource.com/, .

[4] Using Strace - Android Developer Forum. https://source.android.com/docs/core/tests/debug/strace, .

[5] Android Application Logs. https://developer.android.com/reference/android/util/Log/.

[6] APT 29 - Put up your Dukes. https://www.anomali.com/blog/apt-29-put-up-your-dukes.

[7] Arda Computer. https://github.com/Arda-Computer/Arda.

[8] CALDERA. https://www.mitre.org/research/technology-transfer/open-source-software/caldera.

[9] spaCy's English pipeline optimized for CPU (en_core_web_sm-3.5.0). https://github.com/explosion/spacy-models/releases/tag/en_core_web_sm-3.5.0.

[10] Exploit Database. https://www.exploit-db.com/.

[11] Firefox Reality. https://www.meta.com/experiences/firefox-reality/2180252408763702/?srsltid=AfmBOor8RIraMVxITIZSwMPOtelXul5KSYiLQhWSarHrGJQJAiDEHakV.

[12] VR Headset Owners Only Use Their Devices Six Hours a Month on Average (Study). https://variety.com/2019/digital/news/vr-headsets-6-hours-a-month-1203211063/.

[13] Kali Exploit for Quest 2. https://github.com/georgebluff/Kali.

[14] Android Logcat command-line tool. https://developer.android.com/studio/command-line/logcat/.

[15] Meta captures 90% of VR headset market share. https://www.insiderintelligence.com/content/meta-captures-90-of-vr-headset-market-share, .

[16] OpenXR Mobile SDK. https://developer.oculus.com/documentation/native/android/mobile-intro/, .

[17] Metasploit: Basic Discovery. https://www.linux.org/threads/metasploit-basic-discovery.11656/, .

[18] MITRE ATT&CK. https://attack.mitre.org.

[19] UI/Application Exerciser Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey/.

[20] Meta Quest 2. https://www.meta.com/quest/products/quest-2/.

[21] Meta Quest Exploits. https://github.com/QuestEscape/exploit.

[22] Meta Quest Pro. https://www.meta.com/quest/quest-pro/.

[23] NetworkX: Network Analysis in Python. https://networkx.org/.

[24] Computer Security Incident Handling Guide. https://csrc.nist.gov/pubs/sp/800/61/r2/final.

[25] Natural Language Toolkit. https://www.nltk.org/.

[26] The Case Against Root: Why Android Devices Don't Come Rooted. https://www.howtogeek.com/132115/the-case-against-root-why-android-devices-dont-come-rooted/.

[27] NotifiVR. https://sidequestvr.com/app/1442/notifivr.

[28] National Vulnerability Database. https://nvd.nist.gov/vuln.

[29] Oculess: Disable Meta Quest Services. https://github.com/basti564/Oculess, .

[30] Tools for Troubleshooting and Obtaining Log Files. https://developer.oculus.com/resources/log-troubleshooting/, .

[31] OpenXR Software Development Kit. https://github.com/KhronosGroup/OpenXR-SDK-Source/, .

[32] OpenXR Tutorial documentation - Khronos Group. https://openxr-tutorial.com//android/vulkan/4-actions.html#interactions, .

[33] OpenXR - Mixed Reality. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/native/openxr, .

[34] OpenXR Documentation: Procedures and Conventions. https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html, .

[35] OVR Metrics Tool. https://developer.oculus.com/downloads/package/ovr-metrics-tool/.

[36] Android Perfetto Trace tool. https://developer.android.com/studio/command-line/perfetto/.

[37] PhoneSploit Pro. https://github.com/AzeemIdrisi/PhoneSploit-Pro.

[38] PyVis - Interactive network visualizations. https://pyvis.readthedocs.io/en/latest/.

[39] QuestAppVersionSwitcher (QAVS). https://sidequestvr.com/app/5333/questappversionswitcher-qavs.

[40] QuestCraft - Minecraft for XR. https://sidequestvr.com/app/7150/questcraft.

[41] Meta Community Forums: 48-62 FPS acceptable? https://communityforums.atmeta.com/t5/Quest-Development/48-62-FPS-acceptable/td-p/934689.

[42] SideQuest Store. https://sidequestvr.com/.

[43] Windows System Monitor (Sysmon). https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon/.

[44] User Safety in AR/VR: Protecting Adults. https://itif.org/publications/2023/01/17/user-safety-in-ar-vr-protecting-adults/.

[45] Valve Index. https://store.steampowered.com/valveindex, .

[46] SteamVR/Combined Logging - VALVE. https://developer.valvesoftware.com/wiki/SteamVR/Combined_Logging, .

[47] HTC Vive XR Elite. https://www.vive.com/us/product/vive-xr-elite/overview/.

[48] VR Workout. https://github.com/mgschwan/VRWorkout.

[49] PROV-Overview: An Overview of the PROV Family of Documents. http://www.w3.org/TR/prov-overview/.

[50] Wolvic XR Browser. https://www.wolvic.com/en/.

[51] XRStream. https://sidequestvr.com/app/7876/xrstream.

[52] A. Ahmad, S. Lee, and M. Peinado. Hardlog: Practical tamper-proof system auditing using a novel audit device. In *IEEE S&P*, 2022.

[53] M. Backes, S. Bugiel, O. Schranz, P. von Styp-Rekowsky, and S. Weisgerber. Artist: The android runtime instrumentation and security toolkit. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 481–495. IEEE, 2017.

[54] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou. Let sdn be your eyes: Secure forensics in data center networks. In *SENT*, 2014.

[55] P. Casey, I. Baggili, and A. Yarramreddy. Immersive virtual reality attacks and the human joystick. *IEEE TDSC*, 18(2):550–562, 2019.

[56] K. Cheng, J. F. Tian, T. Kohno, and F. Roesner. Exploring user reactions and mental models towards perceptual manipulation attacks in mixed reality. In *USENIX Security*, 2023.

[57] K. Cheng, A. Bhattacharya, M. Lin, J. Lee, A. Kumar, J. F. Tian, T. Kohno, and F. Roesner. When the user is inside the user interface: An empirical study of UI security properties in augmented reality. In *USENIX Security*, 2024.

[58] J. A. De Guzman, K. Thilakarathna, and A. Seneviratne. Security and privacy approaches in mixed reality: A literature survey. *ACM CSUR*, 52(6):1–37, 2019.

[59] M. Du and F. Li. Spell: Streaming parsing of system event logs. In *IEEE ICDM*, pages 859–864. IEEE, 2016.

[60] M. Du, F. Li, G. Zheng, and V. Srikumar. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *ACM CCS*, 2017.

[61] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM TOCS*, 32(2):1–29, 2014.

[62] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song. Enabling efficient cyber threat hunting with cyber threat intelligence. In *ICDE*. IEEE, 2021.

[63] A. Gehani and D. Tariq. SPADE: Support for provenance auditing in distributed environments. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2012.

[64] A. Giaretta. Security and privacy in virtual reality–a literature survey. *arXiv preprint arXiv:2205.00208*, 2022.

[65] H. Guo, S. Yuan, and X. Wu. Logbert: Log anomaly detection via bert. In *IJCNN*, pages 1–8. IEEE, 2021.

[66] W. U. Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *NDSS*, 2018.

[67] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates. NoDoze: Combatting threat alert fatigue with automated provenance triage. In *NDSS*, 2019.

[68] W. U. Hassan, A. Bates, and D. Marino. Tactical provenance analysis for endpoint detection and response systems. In *IEEE S&P*, 2020.

[69] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates. OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis. In *NDSS*, 2020.

[70] M. N. Hossain, J. Wang, R. Sekar, and S. D. Stoller. Dependence-preserving data compaction for scalable forensic analysis. In *USENIX Security*, 2018.

[71] S. Hwang, S. Lee, Y. Kim, and S. Ryu. Bittersweet adb: Attacks and defenses. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 579–584, 2015.

[72] M. A. Inam, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan. Faust: Striking a bargain between forensic auditing's security and throughput. In *ACSAC*, 2022.

[73] M. A. Inam, W. U. Hassan, A. Ahad, A. Bates, R. Tahir, T. Xu, and F. Zaffar. Forensic analysis of configuration-based attacks. In *NDSS*, 2022.

[74] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan. SoK: History is a vast early warning system: Auditing the provenance of system intrusions. In *IEEE S&P*, 2023.

[75] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee. Rain: Refinable attack investigation with on-demand inter-process information flow tracking. In *ACM CCS*, 2017.

[76] S. T. King and P. M. Chen. Backtracking intrusions. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[77] S. Kulal, Z. Li, and X. Tian. Security and privacy in virtual reality: A literature review. *Issues in Information Systems*, 23(2):185–192, 2022.

[78] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, et al. MCI: Modeling-based causality inference in audit logging for attack investigation. In *NDSS*, 2018.

[79] K. H. Lee, X. Zhang, and D. Xu. LogGC: Garbage collecting audit log. In *ACM CCS*, 2013.

[80] K. H. Lee, X. Zhang, and D. Xu. High accuracy attack provenance via binary-based execution partition. In *NDSS*, 2013.

[81] Z. Li, J. Zeng, Y. Chen, and Z. Liang. Attackg: Constructing technique knowledge graph from cyber threat intelligence reports. In *ESORICS*, pages 589–609. Springer, 2022.

[82] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal. Towards a timely causality analysis for enterprise security. In *NDSS*, 2018.

[83] S. Luo, A. Nguyen, H. Farooq, K. Sun, and Z. Yan. Eavesdropping on controller acoustic emanation for keystroke inference attack in virtual reality. In *NDSS*, 2024.

[84] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu. Accurate, low cost and instrumentation-free security audit logging for Windows. In *ACSAC*, 2015.

[85] S. Ma, X. Zhang, and D. Xu. ProTracer: Towards practical provenance tracing by alternating between logging and tainting. In *NDSS*, 2016.

[86] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu. MPI: Multiple perspective attack investigation with semantic aware execution partitioning. In *USENIX Security*, 2017.

[87] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha. Kernel-supported cost-effective audit logging for causality tracking. In *USENIX ATC*, 2018.

[88] Ü. Meteriz-Yıldıran, N. F. Yıldıran, A. Awad, and D. Mohaisen. A keylogging inference attack on air-tapping keyboards in virtual environments. In *IEEE VR*, pages 765–774. IEEE, 2022.

[89] K. Opasiak and W. Mazurczyk. (in) secure android debugging: Security analysis and lessons learned. *Computers & Security*, 2019.

[90] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. Fletcher, A. Miller, and D. Tian. Custos: Practical tamper-evident auditing of operating systems using trusted execution. In *NDSS*, 2020.

[91] K. Pearlman. Virtual reality brings real risks: Are we ready? 2020.

[92] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu. HERCULE: Attack story reconstruction via community discovery on correlated log graph. In *ACSAC*, 2016.

[93] M. U. Rehman, H. Ahmadi, and W. U. Hassan. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *IEEE S&P*, 2024.

[94] A. Romdhana, M. Ceccato, G. C. Georgiu, A. Merlo, and P. Tonella. Cosmo: Code coverage made easier for android. In *IEEE ICST*, 2021.

[95] K. Satvat, R. Gjomemo, and V. Venkatakrishnan. Extractor: Extracting attack behavior from threat reports. In *IEEE EuroS&P*. IEEE, 2021.

[96] Y. Shen and G. Stringhini. Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *USENIX Security*, 2019.

[97] Y. Shen, H. Wen, C. Luo, W. Xu, T. Zhang, W. Hu, and D. Rus. Gaitlock: Protect virtual and augmented reality headsets using gait. *IEEE TDSC*, 16(3):484–497, 2018.

[98] C. Shi, X. Xu, T. Zhang, P. Walker, Y. Wu, J. Liu, N. Saxena, Y. Chen, and J. Yu. Face-mic: inferring live speech and speaker identity via subtle facial dynamics captured by ar/vr motion sensors. In *MobiCom*, pages 478–490, 2021.

[99] C. Slocum, Y. Zhang, N. Abu-Ghazaleh, and J. Chen. Going through the motions: AR/VR keylogging from user head motions. In *USENIX Security*, 2023.

[100] S. Stephenson, B. Pal, S. Fan, E. Fernandes, Y. Zhao, and R. Chatterjee. Sok: Authentication in augmented and virtual reality. In *IEEE Symposium on Security and Privacy*, 2022.

[101] Z. Su, F. H. Shezan, Y. Tian, D. Evans, and S. Heo. Perception hacking for 2d cursorjacking in virtual reality. 2022.

[102] R. Trimananda, H. Le, H. Cui, J. T. Ho, A. Shuba, and A. Markopoulou. OVRseen: Auditing network traffic and privacy policies in oculus VR. In *USENIX Security*, 2022.

[103] W.-J. Tseng, E. Bonnail, M. Mcgill, M. Khamis, E. Lecolinet, S. Huron, and J. Gugenheimer. The dark side of perceptual manipulations in virtual reality. In *CHI*, 2022.

[104] S. Valluripally, A. Gulhane, K. A. Hoque, and P. Calyam. Modeling and defense of social virtual reality attacks inducing cybersickness. *IEEE TDSC*, 2021.

[105] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter. Fear and logging in the internet of things. In *NDSS*, 2018.

[106] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu. Effective real-time android application auditing. In *IEEE Symposium on Security and Privacy*, 2015.

[107] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang. High fidelity data reduction for big data security dependency analyses. In *ACM CCS*, 2016.

[108] C. Yang, G. Yang, A. Gehani, V. Yegneswaran, D. Tariq, and G. Gu. Using provenance patterns to vet sensitive behaviors in android apps. In *International conference on security and privacy in communication systems*, pages 58–77. Springer, 2015.

[109] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen. Uiscope: Accurate, instrumentation-free, and visible attack investigation for gui applications. In *NDSS*, 2020.

[110] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. F. Ciocarlie, V. Yegneswaran, et al. Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation. In *NDSS*, 2021.

[111] X. Yuan, O. Setayeshfar, H. Yan, P. Panage, X. Wei, and K. H. Lee. Droidforensics: Accurate reconstruction of android attacks via multi-layer forensic logging. In *ACM ASIACCS*, 2017.

[112] Y. Zhang, M. Yang, Z. Yang, G. Gu, P. Ning, and B. Zang. Permission use analysis for vetting undesirable behaviors in android apps. *IEEE transactions on information forensics and security*, 9(11):1828–1842, 2014.

[113] Y. Zhang, C. Slocum, J. Chen, and N. Abu-Ghazaleh. It's all in your head (set): Side-channel attacks on ar/vr systems. In *USENIX Security*, 2023.

[114] B. Zhu, J. Li, R. Gu, and L. Wang. An approach to cloud platform log anomaly detection based on natural language processing and lstm. In *Proceedings of the 2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, pages 1–7, 2020.

A DLL Instrumentation

Although our multi-layer logging already captures the most essential forensic events, we also evaluated the performance impact of *native DLL instrumentation* on seven open-source AR/VR applications (apps #9–15). In these experiments, we instrumented selected lifecycle and I/O-related functions (e.g., process creation, file/network I/O) deemed forensically relevant. Appendix A presents the overhead and frame rate drop under both *normal* and *stress-test* workloads[6] for each app. From Table 6, we observe that the *normal* workload overhead averages 29.12%, rising to 45.49% under *stress-test* conditions. Similarly, the average frame rate drop is 27.84% in normal usage and 55.79% in stress tests. No new nodes were identified against ground truths, and hence, these findings indicate that DLL instrumentation provides negligible investigative gains at a substantial performance cost, primarily because we already capture syscall-like data from other layers via less invasive application instrumentation, and off-the-shelf logs. Consequently, we advise users of REALITYCHECK to carefully balance the marginal forensic benefits against the notable impact on runtime overhead and user experience, especially in latency-sensitive AR/VR settings.

Table 6: Native DLL instrumentation overhead and frame rate (F.R.) drop for open-source apps under normal and stress-test workloads.

App #	Overhead (%)		F.R. Drop (%)	
	Normal	Stress	Normal	Stress
9	28.61	51.00	29.12	60.03
10	25.54	39.43	25.87	55.25
11	27.51	41.28	28.45	55.22
12	31.82	46.72	28.84	58.95
13	29.90	45.92	26.50	47.13
14	34.08	50.11	31.12	59.87
15	26.42	43.95	24.98	54.08
Avg	**29.13**	**45.49**	**27.84**	**55.79**

B Generalizability

B.1 Adaptability to Emerging Attacks

REALITYCHECK demonstrates exceptional adaptability to emerging AR/VR threats through its novel five-layered model, specifically designed for flexibility to trace system-wide footprint of emerging attacks. The only thing that the user needs to do is that, should a new entity emerge (as shown in Figure 4), they need to annotate unstructured logs and simply fine-tune the NER model, as REALITYCHECK's design facilitates the integration of novel attack attributes with these minimal adjustments. More importantly, the REALITYCHECK's core mechanisms, including edge recovery, execution partitioning, and graph pruning techniques, are inherently designed to be generalizable to new threats without necessitating modifications since these methods are unsupervised and do not require training. As shown in §6.1, Attacks 3-4 and 19 emerged in 2024, after we prototyped REALITYCHECK for Meta Quest 2.

[6] Stress testing was performed using the monkey toolkit [19].

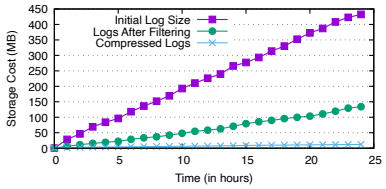


Figure 9: Storage consumed by REALITYCHECK over time.

Table 7: Effectiveness of REALITYCHECK in tracing the provenance of different attack scenarios when tested on HTC Vive [47]. G.T.: Ground Truth; RC: REALITYCHECK; EP: Execution Partitioning; GP : Graph Pruning; V: Vertices; E: Edges; Cov.: Coverage.

	Investigation Effectiveness				NLP Effectiveness			
	GT	RC	RC + EP & GP		TP/FP/FN	Prec.	Rec.	F1
	V/E	V/E	V/E	Cov.				
3	32/37	104/122	32/36	0.99	68/2/0	0.97	1.0	0.99
4	31/36	98/120	31/36	1.0	67/3/0	0.96	1.0	0.98
19	16/16	57/59	16/16	1.0	32/1/0	0.97	1.0	0.98

Table 8: Compatibility of REALITYCHECK across devices in comparison to the baseline Meta Quest 2. • represents same logs, ○ represents different logs. The last column shows device compatibility with REALITYCHECK.

Device	Layer					Compatibility
	1	2	3	4	5	
Meta Quest Pro [22]	•	•	•	•	•	✓
HTC Vive [47]	•	•	○	•	•	✓
Valve Index [45]	•	•	○	•	•	✓

Table 3 shows that REALITYCHECK was able to effectively investigate these attacks, all with perfect coverage of 1.0. This proves REALITYCHECK's adaptability to emerging threats.

B.2 Generalizability to Other AR/VR Devices

The generalizability of REALITYCHECK to other AR/VR devices depends on the presence of log sources within the device layers illustrated in Figure 1. As seen in Table 8, REALITYCHECK has been verified to be compatible with multiple popular devices, including HTC Vive and Valve Index (with minor adjustments for Steam Logs), and a detailed mapping is provided in the Appendix (§D.0.1). Additionally, REALITYCHECK was tested on the HTC Vive XR Elite [47] to validate its effectiveness against three newly emerged 2024 attacks. Only the parser for SDK Layer logs needed modification (D.0.1), without requiring extra data labeling for the NLP component. Table 7 reports near-perfect coverage, precision, recall, and F1-scores, demonstrating REALITYCHECK 's capacity to work with AR/VR devices beyond Meta Quest 2.

C Storage Overhead

In this appendix, we provide additional results that showcase the storage growth associated with REALITYCHECK. Figure 9 shows storage growth over time from hourly automated scripts simulating use when REALITYCHECK is deployed on Meta Quest 2. Over 24 hours, raw logs were 432MB. With partitioning and pruning, this was reduced to 134MB (12MB compressed). We optimized Perfetto logs by extracting only essential `sched_switch` info (§5) and purging surplus logs every 10 minutes. Given an average monthly headset use of 6 hours [12], our system's overhead is minimal for users.

D Detailed Algorithm for Bytecode Instrumentation

The BYTECODEINSTRUMENTOR algorithm (Algorithm 2) plays a crucial role in enhancing our AR/VR-aware execution partitioning technique by instrumenting specific functions within an application file, which are needed for identifying execution units effectively. How these functions are identified is explained in §5.4. Given an application file and the list $L_{Functions}$, the algorithm proceeds by iterating over each method m in the application's bytecode. For each method, it checks whether the method is included in $L_{Functions}$. If so, the body b of method m is retrieved, and the unit chain U—which represents sequences of bytecode instructions—is extracted from b. The algorithm then initializes a new unit chain U', where it will store modified (instrumented) units. For each unit u in U, the algorithm checks if u is an invoke statement and if the invoked method is within $L_{Functions}$. If both conditions are met, it generates and appends logging statements to U', which capture method invocations and their arguments, thereby preparing a detailed logging message. After processing all units in U, the instrumented unit chain U' replaces the original U in b. Following the modification, the bytecode is validated to ensure it remains executable and conforms to bytecode standards. The instrumented application is then constructed by running transformation packs via a transformer module, which applies the bytecode modifications across the entire application file.

```
public static void OnApplicationPause(boolean):
paused = @parameter0: boolean
// The added logging statements:
$r2 = <java.lang.System: java.io.PrintStream out>
argLocal_0 = paused
$r4 = virtualinvoke argLocal_0.<java.lang.Boolean: java.lang.String toString()
$r5 = "Beginning of method OnApplicationPause with arguments [" + $r4 + "]"
virtualinvoke $r2.<java.io.PrintStream: void println(java.lang.String)>($r5)
```

Figure 10: Jimple code after instrumentation for the `OnApplicationPause` method.

Unity and Unreal Engine provide lifecycle management methods—such as Unity's `OnApplicationPause`, `OnApplicationFocus`, `OnApplicationQuit`, and Unreal's `ApplicationWillEnterBackgroundDelegate`, `ApplicationHasEnteredForegroundDelegate`, `ApplicationWillTerminateDelegate`—that mirror the functionality of OpenXR's `xrPollEvent` and Android's `onPause`/`onResume`. By instrumenting these methods, developers can log and analyze key lifecycle transitions similar

Algorithm 2: BYTECODEINSTRUMENTOR

Inputs : Application file, List of target functions to instrument $L_{Functions}$.
Output : Instrumented Application file.

```
foreach method m in the Application bytecode file do
    if m ∈ L_Functions then
        Retrieve method body b of m and extract unit chain U from b;
        Initialize new unit chain U' for instrumented units (or statements);
        foreach unit u ∈ U do
            if u is an invoke statement and InvokedMethod(u) ∈ L_Functions
              then
                logStmts ← GenerateLogStmts(u);
                Extract method name and arguments from u;
                Prepare logging message;
                U'.add(logStmts);
            U'.add(u);
        Replace U in b with U';
        Validate b;
Transformer.v().runPacks();
Transformer.v().writeOutput();
return InstrumentedApplication
```

to how they might with OpenXR. This approach ensures a detailed and consistent view of application behavior across different engines and SDKs, as demonstrated by the Jimple code example for Unity's `OnApplicationPause` method (Figure 10).

D.0.1 Generalizability: Mapping Oculus and Steam logs

The generalizability of REALITYCHECK across different AR/VR devices is crucial. While the prototype focuses on the Meta Quest 2 (using Oculus logs), it can be adapted for devices that rely on Steam logs, such as HTC Vive XR Elite [47] and Valve Index [45]. As noted in §B.2, the only significant variation lies in the SDK-layer logs for these popular hybrid AR/VR devices. This appendix details a mapping for these logs, highlighting key characteristics from Figure 5: process details (C10), device id (C12), and device event type (C13)[7].

Process Details (C10). In Oculus logs, process details are typically captured under the tag `[Process]` with entries detailing the process name, PID, and other relevant metadata. In Steam logs, the equivalent information can be found under the tag `[AppProcess]`.

Device ID (C12). Oculus logs identify devices using a unique identifier under the tag `[Device_ID]`. Steam logs, on the other hand, use the tag `[HardwareID]` to capture the same information.

Device Event Type (C13). For Oculus logs, device events such as connection or disconnection are logged under the tag `[ETW_USB_EVENT]` or `[ETW_TCP_EVENT]`. In Steam logs, similar events are captured under the tag `[DeviceEvent]`.

The mapping provided above showcases the similarities between Oculus and Steam logs, emphasizing that the adaptation of REALITYCHECK from one platform to another is straightforward. By understanding the structure and tags used in each logging system, we can easily extend REALITYCHECK's capabilities to cater to a broader range of AR/VR devices.

[7]The Steam logs were collected from HTC Vive XR Elite [47]. They are similar in nature to what we observed with Valve Index [45].

E Existing AR/VR Attacks

Eavesdropping attacks involve malicious apps installed on the victim's AR/VR device, capturing motion sensor data and transmitting it to a remote attacker to reconstruct sensitive user information, such as passwords [98]. Human Joystick attacks occur when a malicious application alters the user's perception and boundary details to manipulate the user into a predefined area by changing the device's configuration settings [103]. Chaperone attacks involve malicious applications that modify AR/VR borders by altering device configurations, causing virtual areas to appear larger or smaller to the immersed user [55]. Overlay attacks superimpose unwanted images, videos, or content onto a user's AR/VR view via a malicious background application [55]. Cybersickness attacks induce dizziness and disorientation in the user through device haptics or digital objects thrown at the user, potentially triggering epilepsy via malicious applications installed on the AR/VR device [104]. Key-logging Interface attacks introduce malware onto the victim's AR/VR headset, allowing the attacker to access the victim's hand-tracking data through the headset's API [88]. Camera Stream and Tracking Exfiltration attacks leverage background applications running on the AR/VR device, enabling a malicious program to retrieve camera streams and tracking details without displaying an image or requiring a specific scene [55, 98].

We further map these AR/VR attacks according to MITRE TTPs [18] in Table 9. This taxonomy presents a comprehensive attack matrix, classifying these attacks and offering valuable insights into the AR/VR threat landscape.

F AR/VR Attack Literature Review

For our attack modelling, we embarked on an extensive review of academic literature using DBLP and Google Scholar, pertinent open-source projects hosted on GitHub [1, 13] and CVE databases [10, 28], spanning the previous five years. Our research was underpinned by a carefully selected set of keywords including but not limited to "AR/VR security", "XR security", "AR/VR exploits", "AR/VR exploits", "AR/VR vulnerabilities", "XR vulnerabilities", "AR/VR attacks", "XR attacks", "immersive attacks", "AR/VR device configuration attacks", "XR device configuration attacks", "user perception manipulation", and "mixed reality attacks". This in-depth investigation led to the identification of 23 distinct attacks targeting various facets of hybrid AR/VR devices like the Meta Quest, HTC Vive, and Valve Index. With these identified attack vectors in mind, we tailored our log collection strategy to capture evidence of these specific threats, ensuring a relevant, comprehensive, and efficient data acquisition for our analysis.

Table 9: Attack matrix for the AR/VR ecosystem.

Attack Name	Tactics	Techniques	Procedures
Perception Manipulation	Overlay	Overlay partial screen with digital object [55, 103]	Use a malicious application to overlay a small and invisible digital object onto the screen. The attack was simulated using the code provided by the authors.
		Overlay entire screen with digital object [55, 103]	Use a malicious application to overlay an invisible digital object onto the entire screen. The attack was simulated using the code provided by the authors.
		Object-in-the-middle attack [57]	Overlay an invisible object on top of a visible virtual object, such as a keyboard, to record user input without their knowledge. The attack was simulated using the Unity assets provided by the authors.
		Object erasure attack [57]	Employ fully transparent meshes to render targeted AR content invisible, thereby distorting the user's reality by erasing vital information. The attack was simulated using the Unity assets provided by the authors.
	Human Joystick Attack	Alter virtual boundary coordinates [55, 103]	Alter the HMD headset's device configurations through a remote access trojan on an infected PC, and then flas the configurations via an android toolkit to distract/disorient the victim.
	Chaperone Attack	Alter virtual boundary sensitivity [55, 103]	Modify the headset's VE sensitivity using an android toolkit by setting it to a low value casing collision with the VE boundary. The attack was simulated using the techniques described in the paper since no official source code was available.
Eavesdropping Attack	Discovery	Software discovery [13]	Use Metasploit to identify the software installed on the target device. The repository providing setup for this attack is cited.
		System information discovery [13]	Use malicious payload to scan the target device for system information. The repository providing setup for this attack is cited.
		System service discovery [13]	Use a remote access payload to connect to the headset and enumerate running services. The repository providing setup for this attack is cited.
		Network configuration discovery [89]	Use PC connected to the headset to retrieve its network configuration via Nmap. The attack was simulated using the techniques described in the paper since no official source code was available.
	Collection	Access data from/using foreground applications [83, 99]	Use side-channels to access and exfiltrate data from the HMD device. The attack was simulated using the techniques described in the paper since no official source code was available.
		Immersive browser session hijacking [37]	Use malicious payload to pull browser cookies from the HMD device and use them to hijack a browser session.
		Clipboard data [89]	Use a remote access payload to capture and exfiltrate clipboard data from the target device, or use an android toolkit to exfiltrate this data via the connected PC. The attack was simulated using the techniques described in the paper since no official source code was available.
	Exfiltration	Automated exfiltration [13, 37]	Use Metasploit post-exploitation module `reverse_tcp` to install backdoor and exfiltrate data from the target device. The repository providing setup for this attack is cited.
		Exfiltration via endpoint system [91]	The attack utilized `dumpsys` to transfer data from an HMD device to a PC via an Android toolkit, followed by exfiltration through the connected PC. The simulation was conducted using techniques from the referenced presentation, due to the unavailability of official source code. The tactic, named by us and pertaining to endpoint data exfiltration, draws inspiration from the cited work as outlined on Page 11 of the source.
Physical Harm	Cybersickness	Dizziness [104]	Send excessive haptic events via an application installed on the headset or via the connect PC through an android toolkit. The attack was simulated using the techniques described in the paper since no official source code was available.
		Force play audio [29]	Use malicious payload to force play audio in the background. The potentially malicious app to run the attack is cited.
		Photosensitive epilepsy [104]	Install an app on the HMD that includes flashing or strobing lights or toggle brightness via an android toolkit. The attack was simulated using the techniques described in the paper since no official source code was available.
Key-logging Interface Attack	Credential Access	Input capture [83]	Use a malicious app to record user input, such as controller coordinates. The attack was simulated using the techniques described in the paper since no official source code was available.
		Retrieve credentials [57]	Retrieve unsecured credentials from the device, such as plain text password files via the connected PC or through a malicious payload on the HMD that overlays a digital object over a user's keyboard. The attack was simulated using the code provided by the authors.
Defense Evasion	File Deletion	Delete files and applications with write acces [71, 89]	Use file deletion techniques to remove files and applications on the headset that have read/write access. The attack was simulated using the techniques described in the paper since no official source code was available.
	Data Destruction	Permanently delete user data [89]	Use data destruction techniques to permanently delete user data on the headset via malicious payload or through the connected PC via an android toolkit. The attack was simulated using the techniques described in the paper since no official source code was available.
AR/VR Device Exploitation	Device Impact	Service stop [29]	Disable Facebook services on the headset via malicious application on the HMD. The potentially malicious app to run the attack is cited.
			Disable device telemetry services via malicious application on the HMD. The potentially malicious app to run the attack is cited.
			Force stop or disable updates via malicious application on the HMD. The potentially malicious app to run the attack is cited.
		Account access removal [29]	Remove Facebook Accounts from Device via malicious application on the HMD. The potentially malicious app to run the attack is cited.
		Force shutdown / reboot [37, 71]	Execute force shutdown intent via the connected PC. The toolkit used to run the attack is cited.