Virtual Reality, *Real* Problems:
A Longitudinal Security Analysis of VR Firmware

Vamsi Shankar Simhadri
Department of Computer Science
George Mason University
Fairfax, VA, United States
vsimhadr@gmu.edu

Yichang Xiong
Department of Computer Science
George Mason University
Fairfax, VA, United States
yxiong2@gmu.edu

Habiba Farrukh
Department of Computer Science
University of California, Irvine
Irvine, CA, United States
habibaf@uci.edu

Xiaokuan Zhang
Department of Computer Science
George Mason University
Fairfax, VA, United States
xiaokuan@gmu.edu

Abstract

Virtual Reality (VR) technology is rapidly growing in recent years. VR devices such as Meta Quest 3 utilize numerous sensors to collect users' data to provide an immersive experience. Due to the extensive data collection and the immersive nature, the security of VR devices is paramount. Leading VR devices often adopt and customize Android systems, which makes them susceptible to both Android-based vulnerabilities and new issues introduced by VR-specific customizations (e.g., system services to support continuous head and hand tracking). While prior work has extensively examined the security properties of the Android software stack, how these security properties hold for VR systems remains unexplored.

In this paper, we present the *first* comprehensive security analysis of VR firmware. We collect over 300 versions of VR firmware from two major vendors, Quest and Pico, and perform a longitudinal analysis across the kernel layer, the system binary and library layer, and the application layer. We have identified several security issues in these VR firmware, including missing kernel-level security features, insufficient binary hardening, inconsistent permission enforcement, and inadequate SELinux policy enforcement. Based on our findings, we synthesize recommendations for VR vendors to improve security and trust for VR devices. This paper will act as an important security resource for VR developers, users, and vendors, and will also direct future advancements in secure VR ecosystem.

CCS Concepts

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

Keywords

Virtual Reality; Firmware Analysis

1 Introduction

Virtual Reality (VR) technology is a rapidly growing trend with a projected market value of $22 billion in 2025 [90]. One of the industry's leaders, Meta (formerly known as Facebook), has sold approximately 20 million VR headsets as of March 2023 [82, 98] and invested billions in its VR venture. Apple recently launched Vision Pro [11], a new mixed reality device, and Google is expected to release the AndroidXR operating system in 2025 [42].

Despite its widespread adoption, VR technology introduces critical security and privacy challenges. The extensive collection and processing of personal data by VR devices, including biometric identifiers, behavioral patterns, and environmental information, raises significant privacy concerns [54, 97, 102]. The compromise of such sensitive data could lead to severe consequences, including identity theft, unauthorized system access, and invasion of personal privacy. Beyond data protection, VR systems present unique safety challenges [34, 51] due to their immersive nature, which completely obscures users' perception of their physical surroundings [16]. To mitigate potential physical harm, VR devices implement runtime safeguards, such as virtual boundaries [68], to prevent accidents like dangerous falls and collisions. Yet, these safety mechanisms are mediated by the VR device's operating system and could be rendered ineffective if the software stack is compromised, leading to far-reaching implications for end-user well-being.

These risks underscore the importance of robust security in the software that powers VR platforms. Leading VR platforms such as Meta Quest and Pico have strategically adopted customized Android-based operating systems, leveraging Android's robust open-source architecture as a foundation. These manufacturers undertake sophisticated customization processes to seamlessly integrate proprietary system software with the base operating system, resulting in device-specific **firmware** that powers their cutting-edge products. While the core Android framework maintains its

open-source nature, the customized firmware implementations remain proprietary and closed-source, creating significant barriers to transparency and independent security assessment.

Operating as critical downstream components in the software supply chain, these modified firmware systems face a dual security challenge: they inherit Android's inherent vulnerabilities while introducing novel security considerations through extensive customization. The Android Security Bulletin [39] consistently reveals numerous critical security vulnerabilities requiring immediate patching, yet downstream device manufacturers face substantial challenges in implementing these crucial updates in a timely manner [109, 112]. This concern is further amplified by extensive research demonstrating that vendor-specific Android customizations frequently introduce additional security vulnerabilities [4, 28, 53, 60, 80, 100, 104, 113]. These risks are significantly amplified in VR systems, where firmware must support continuous, high-bandwidth interaction with complex sensors like eye trackers, and depth sensors as well as scene and spatial understanding pipelines. Supporting such capabilities often necessitates architectural changes to the Android-based firmware, which may introduce novel vulnerabilities.

While prior security research on VR systems [5, 50, 55, 59, 62, 63, 71–73, 76, 78, 79, 87, 91, 94, 111] has primarily focused on application-level threats and user privacy concerns, such deanonymization via sensor data analysis, behavior profiling, and keystroke inference through side-channel attacks, the security of underlying VR firmware powering these systems remains largely unexplored. As VR technology rapidly transitions from an emerging technology to mainstream adoption, there is a critical need to systematically investigate potential security vulnerabilities in VR firmware. This research gap is particularly concerning given VR's expanding role in sensitive domains like military training, healthcare, and enterprise environments, where firmware vulnerabilities could potentially compromise not only individual user safety but also organizational security.

In this paper, we seek to address this research gap by analyzing how VR vendors adopt security measures to protect VR firmware. Android has evolved a rich set of protections, including kernel-level mitigations, binary hardening, access control policies and SELinux-based isolation [7, 8, 64, 88] to protect against various vulnerabilities. However, it is unclear whether these defenses are preserved, degraded, or bypassed in VR firmware, where performance demands and proprietary customizations are intense.

Challenges. Analyzing the firmware security for VR platforms, however, presents several challenges. First, unlike Android smartphones, where root access is readily available, VR devices are designed to be tamper-resistant and do not support root access. Second, firmware security-related configurations, such as kernel configurations, SEPolicies, or permission settings, are spread across multiple image layers and file paths in VR firmware. Lastly, VR platforms lack standardized firmware structure. For instance, Meta uses `payload.bin`, while Pico uses Brotli-compressed partitions; some devices consolidate firmware customizations in `system.img`, others split them across `system.img`, `vendor.img`, and `odm.img`. This variability creates a fragmented firmware landscape where firmware analysis tools must be adapted for each target.

To address these challenges and fill the current research gap, we present the *first* comprehensive, longitudinal investigation into VR firmware security. Our study employs a systematic approach to analyze VR firmware security across multiple architectural layers (kernel layer, binary layer, and application layer) through rigorous differential analysis against the Android Open Source Project (AOSP) baseline in relevant cases. We have assembled an extensive dataset comprising over 300 firmware versions from two leading VR manufacturers, Meta and Pico, enabling us to conduct a thorough longitudinal analysis. From a high-level, we unzip each firmware, and extract image files that are important for our study; then, we mount those images locally to extract necessary files (e.g., kernel binaries, preinstalled apps), and perform analysis.

Findings. Our analysis of VR firmware reveals systemic security weaknesses across multiple layers of the software stack. At the kernel level, VR devices frequently suffer from misconfigurations and delayed adoption of up-to-date Linux versions, leading to privilege escalation attacks. Similarly, binary hardening is inconsistently applied, with key protections like Control Flow Integrity (CFI) and Fortify Source often absent. At the application level, improper security flag usage and flawed permission handling expose VR devices to risks such as unauthorized data backups, man-in-the-middle (MITM) attacks, and permission leaks. Moreover, vendors frequently modify SEPolicies and leave system properties exposed to untrusted apps. Collectively, these findings suggest that despite a growing market presence, VR devices lag behind in adopting standard mobile security practices.

Contributions. This paper makes the following contributions:

- We present the first longitudinal, multi-layer investigation into VR firmware security. Our work complements prior studies on vendor customizations and firmware vulnerabilities for Android and IoT devices, but we focus on VR, where distinct risks stem from continuous real-time sensor input, immersive runtime dependencies, and proprietary firmware design.

- We develop a set of tools to extract and analyze VR firmware across multiple vendors and hardware versions, which are designed for fragmented and vendor-specific VR firmware formats. The tools will be open-sourced to facilitate future research.

- We present several important findings regarding the security of VR firmware, including missing kernel-level security features, insufficient binary hardening, inconsistent permission enforcement, and inadequate SELinux policy enforcement. Based on our findings, we synthesize recommendations for VR vendors to improve security and trust for VR devices.

Ethics consideration. The firmware dataset used in this paper was collected from public resources. We have disclosed our findings with Meta and Pico teams, and their responses are discussed in §8.

Artifacts. Our code and data are available at: https://github.com/SECSAT-LAB-GMU/VR-Firmware.

2 Background

2.1 VR Device Types

Virtual Reality (VR) offers immersive 3D experiences via advanced head-mounted displays (HMDs) [86]. The combination of market
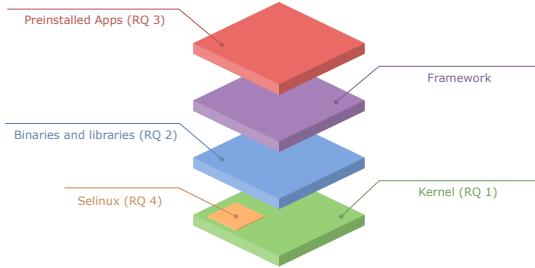
Fig. 1: VR Firmware

opportunities and technology innovation has driven significant investments by major tech companies in proprietary VR platforms [12]. VR systems are categorized into four types based on architecture: (1) Basic Optical VR [40] (e.g., Google Cardboard), using passive optical elements like precision convex lenses; (2) Smartphone-tethered VR [29] (e.g., Samsung Gear VR), using mobile devices with tracking sensors; (3) PC-tethered VR [17] (e.g., Valve Index), utilizing high-performance PCs for graphics and physics; and (4) Standalone VR [70] (e.g., Meta Quest 3), combining processing units, displays, and sensors into a self-contained platform. This paper focuses on standalone VR devices as the leading market segment and complete computational systems with comprehensive firmware. We also only focus on Android-based devices such as Meta Quest.

2.2 VR Firmware

Most VR devices run customized Android firmware containing essential components, policies, and configurations for proper functioning. This firmware includes executables, libraries, pre-installed apps, and scripts for system setup. All use at least Android 10, except Oculus Quest, which started with Android 7.1.1 and later upgraded to Android 10. Manufacturers enhance firmware with VR-specific binaries, services, and applications, adjusting functionalities. Customizations include integrating kernel components like VR drivers and refining security and performance for immersive experiences. At a high level, VR firmware follows the following structure [9] (shown in Fig. 1), with vendors frequently modifying these components to meet their specific needs.

Kernel Layer. The kernel serves as the foundation of the VR device, providing essential system functionalities. The VR kernel layer consists of the core Android kernel with VR-specific modifications. These include VR-specific drivers, such as display drivers, GPU drivers, and sensor drivers, which ensure performance and responsiveness. Within the firmware, the kernel binary is stored in `boot.img`.

System Binaries and Libraries Layer. Binaries and libraries within the firmware are essential system components that provide pre-compiled functions and services required by both the operating system and applications. These components are crucial for system stability and performance. However, they are also a common source of critical security vulnerabilities [80], as many are developed using unsafe programming languages like C and C++, which can introduce memory-related vulnerabilities.

Framework Layer. The Android framework acts as an intermediary layer between applications and the underlying system. It provides APIs that allow apps to interact with hardware sensors, networking, graphics, and user interface components. The framework translates app requests into system calls, enabling communication with binaries, libraries, and the Linux kernel.

Application Layer. Apps interact with the Android framework to access system resources like sensors, network, and storage. The apps include system apps, pre-installed apps, and third-party apps. System apps are essential applications that provide system-level functionalities, while pre-installed apps are the apps provided by the vendors.

2.3 Android Customization

Developing an Android-based system involves a multi-stage process that begins with the strategic selection of an Android version as the foundational base image. This critical decision determines the SDK level and technological capabilities available to the system. Following version selection, manufacturers systematically fork the corresponding branch from the Android Open Source Project (AOSP) repositories to establish their development baseline. The customization phase involves comprehensive system-wide modifications that span both user-space and kernel-space components. Vendors integrate proprietary binaries, services, and core system modifications to achieve their desired functionality. At the kernel level, they implement essential drivers for custom peripherals and make strategic adjustments to other components (e.g., security policies and initialization scripts) to optimize system performance and security. For VR devices specifically, the extensive array of specialized sensors and peripherals requires substantial kernel-level customization. This includes implementing custom device drivers and developing kernel support infrastructure to enable seamless interaction between userspace APIs and VR-specific hardware components (*e.g.*, eye-tracking sensors).

2.4 Android Compliance Check

Android enforces a strict compliance framework to ensure device compatibility and security. Central to this is the Android Compatibility Definition Document (CDD) [10], which outlines the technical specifications for *Android-compatible* devices. For each release, Google provides a version-specific CDD integrating SDK API documentation and implementation guidelines to ensure consistency. Manufacturers must pass a detailed validation process checking both AOSP compatibility and security to gain certification. The Compatibility Test Suite (CTS) [43] is the main assessment tool, testing adherence to CDD standards, ensuring devices meet Android's compatibility and security requirements.

The CDD framework plays a crucial role in balancing innovation with standardization by establishing clear boundaries for vendor customization. While manufacturers retain the flexibility to enhance and differentiate their Android-based products through custom modifications, these adaptations must conform to CDD guidelines to preserve compatibility. While Android-based VR devices are not mandated to obtain CDD certification, adherence to these established standards provides a critical framework for evaluating device security posture. For example, the CDD specifications outline essential kernel-level security requirements for specific Android versions, establishing baseline protections that are

Device	Version	Size	Ave. Binary Per Version	Ave. App Per Version	Date Range	Android version
Quest	54	100GB	1956	103	June 2019 - December 2023	Android 7.1.1, 10
Quest 2	151	350GB	3049	117	September 2020 - December 2024	Android 10, 12
Quest 3	37	150GB	3351	131	September 2023 - December 2024	Android 12
Quest Pro	79	170GB	3290	127	March 2023 - December 2024	Android 10, 12
Pico Neo 3	4	14GB	4132	181	May 2021 - December 2024	Android 10
Pico 4	4	14GB	4350	176	October 2022 - December 2024	Android 10

Table 1: VR Firmware Dataset

fundamental to maintaining robust security across VR systems. As there is no such standard for VR devices yet, the requirements from Android CDD serve as valuable reference points for assessing the security maturity of VR implementations.

2.5 Android Security Features

Modern Android systems implement a range of security mechanisms across multiple layers of the software stack. At the kernel level, Android has implemented defenses like kernel page table isolation and KASLR to protect the kernel space. Similarly, Android enforces binary hardening techniques [80, 105] on system binaries such as stack canaries, position-independent executables (PIE), and full RELRO to mitigate various security vulnerabilities. Moreover, Android relies on SELinux as a mandatory access control framework to confine system processes and limit privilege escalation through fine-grained policies. These mechanisms form the foundation of Android's defense-in-depth strategy. Although modern VR devices rely on customized Android versions, they often deviate from the standard Android release process and may include vendor-specific modifications, custom builds, or legacy components. In this work, we focus on understanding whether these well-established security practices are faithfully translated into VR firmware across device generations or whether gaps exist that could expose VR devices to additional security risks. We studied existing Android firmware analysis works [28, 30, 32, 33, 46–48, 64, 80, 92] and identified critical security features in Android, which motivates our study.

3 Overview

In this section, we first present the research questions guiding our study. We then describe the VR firmware dataset we curated. Finally, we detail the our analysis pipeline used to process and extract relevant information from the firmware.

3.1 Research Goals

As VR platforms continue to grow in complexity and adoption, the firmware powering these devices becomes a critical component in VR systems' security and privacy guarantees. Unlike traditional mobile platforms, VR firmware vulnerabilities can compromise both privacy and users' physical safety. Misplaced Guardian boundaries, camera spoofing, or corrupted sensor data can cause disorientation and injury. Moreover, VR devices process continuous sensor data (e.g., eye/face/body trackers), making kernel-level misconfigurations and weak SEPolicy rules particularly dangerous. However, the security of VR firmware and the implications of its design decisions over time remain largely unexplored. To systematically analyze its security posture, we focus on the following research questions to

understand the security of different layers of the firmware stack (Fig. 1):

- **RQ1:** What kernel-level security features are missing or inconsistently deployed in VR firmware?

- **RQ2:** Are the binaries in VR firmware properly hardened with appropriate defenses?

- **RQ3:** What privileges and security controls are applied to pre-installed applications in VR firmware?

- **RQ4:** How are SELinux policies enforced across firmware versions and device generations?

We perform a longitudinal analysis across firmware updates and device generations to identify regressions or improvements, and better understand how vendors evolve their security measures for VR devices.

3.2 VR Firmware Dataset

Our study focuses on standalone VR headsets running Android-based firmware. We compiled a list of major standalone VR vendors and excluded those without publicly available firmware (e.g., HTC-Vive and Varjo). To conduct our longitudinal analysis, we curated a comprehensive dataset of firmware images from two leading VR device vendors: Meta and Pico. Meta and Pico were selected because their firmware is available, and the two vendors represent over 82% of the VR headset market [18], making them highly representative of the consumer VR landscape. Specifically, we target the following standalone VR devices: Meta Quest 1, Quest 2, Quest 3/3S, Quest Pro, Pico Neo 3, and Pico 4.

To collect firmware images for these devices, we searched various online sources, including forums, developer communities and archival websites[2, 13, 24, 31, 35, 61, 66]. For Meta Quest devices, we identified a repository [31] that hosts a comprehensive archive of firmware updates across all Quest product lines. For Pico devices, we collected firmware directly from official update channels [61] and manually retrieved older firmware versions using the Wayback Machine [13], which allowed us to access historical download links that were no longer publicly listed. We excluded devices that have been available only for a limited period. For instance, Meta Quest 3S was released in October 2024 and the firmwares are only available for a two month duration (Oct - Dec 2024) at the time of our data collection.

Our final dataset includes over 300 distinct firmware versions across the six VR devices. The firmware images span a date range from March 2019 to December 2024, and total more than 700GB in size. Table 1 summarizes the number of versions per device, their average binary and app counts, and the underlying Android base versions. We observed a clear progression in the Android base versions across different VR device generations. The original
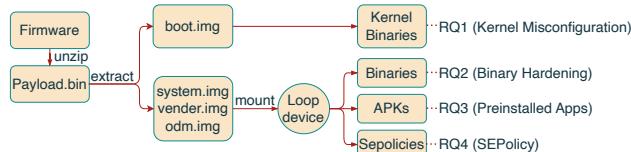
Fig. 2: High-level Approach

Quest device initially used Android 7.1.1 as its foundation before transitioning to Android 10 in subsequent updates. Quest 2 and Quest Pro devices were developed on Android 10 and later received major version updates to Android 12. The Quest 3 was developed on Android 12, representing the most up-to-date Android adoption in Quest devices. In contrast, Pico has maintained Android 10 as the base operating system across their device lineup, demonstrating a more conservative approach to Android version adoption. This longitudinal dataset provides a strong foundation for analyzing security practices across both vendors and multiple generations of Android-based VR devices.

3.3 Firmware Image Analysis

Fig. 2 illustrates our high-level workflow used to extract, process, and analyze artifacts from each VR firmware image. Our goal is to retrieve kernel binaries, user-space executables, pre-installed applications, and SELinux policies from each image to support the security analyses defined by our research questions.

We analyze each firmware image via a three-phase approach. First, we decompress the firmware packages. VR firmware images are typically distributed as `.zip` archives. We use standard tools such as `unzip` to extract the contained payloads (i.e., `payload.bin` and Brotli compressed files). Next, we extract the relevant system partition images. For Meta firmware, we use a payload extraction tool [89] to extract key image files, including `boot.img`, `system.img`, `vendor.img`, and `odm.img`. For Pico firmware, we use `brotli` [6] to retrive the decompressed Brotli (`.br`) files which contain these partition images. Lastly, we mount these images to extract the relevant artifacts for our analysis using standard Linux loop device and extraction tools (e.g., `boot.img` editor [36]). Specifically, we extract the kernel binaries from `boot.img` for RQ1 and extract native binaries, pre-installed applications, and SELinux policies from `system.img`, `vendor.img`, and `odm.img` one by one for answering RQ2, RQ3 and RQ4. To identify the Android version, we parse the `build.prop` configuration file located in the mounted `system` partition. For Meta devices, we use the `ro.build.id` field, while for Pico devices we rely on `ro.system.build.version.release` and `ro.system.build.version.sdk`. Table 8 in Appendix of [85] summarizes the key paths and properties we use to extract security-relevant data from each firmware image. This pipeline enables scalable, automated analysis across hundreds of firmware versions and serves as the foundation for the longitudinal firmware analysis presented in the remainder of the paper.

4 RQ1: Kernel Misconfiguration

4.1 Motivation

The Android Compatibility Definition Document (CDD) [10] establishes critical security requirements for kernel compilation, mandating specific security features to mitigate potential vulnerabilities. Non-compliance with these requirements can lead to severe security implications, as demonstrated by CVE-2018-9568 [19], which enabled privilege escalation attacks on Android systems through kernel misconfiguration. Given that VR devices utilize customized Android kernels, they inherit similar vulnerability patterns. As one concrete example, CVE-2018-9568 was successfully exploited on Oculus Quest devices [38] for privilege escalation, illustrating how security weaknesses can propagate across the Android ecosystem to VR platforms.

In response to such vulnerabilities, the Android Open Source Project (AOSP) has implemented robust kernel-level security mitigations [41], such as Kernel Address Space Layout Randomization (KASLR) and Privileged Access Never (PAN). However, prior research works [112] have shown that nearly half of the CVEs are patched on OEM devices roughly 200 days or more after the initial patch is publicly committed in the upstream. This significant patch deployment latency, combined with VR devices' shared kernel architecture with Android, underscores the critical importance of systematic kernel security misconfiguration analysis in VR platforms. Such analysis is essential to ensure VR devices maintain parity with Android's evolving security standards.

4.2 Methodology

For identifying the mitigations present in the kernel, we need to extract the kernel configuration file (`.config`) from the kernel binary. To do so, we used the *extract-ikconfig* script [37]. Unlike existing studies that face difficulties in extracting kernel configurations [80, 105], we were able to extract all the `.config` files from the kernel binaries because all VR firmware were compiled with the `CONFIG_IKCONFIG` flag [3]; This config flag is a kernel configuration option in the Linux kernel that allows the kernel to include its own configuration file as part of the compiled kernel image, and it has been required since Android 8.0. As most of the VR firmware in our dataset uses an Android version higher than 8.0 and the ones that have used Android 7.1.1 (from Quest) also have this flag enabled, we were able to extract kernel configurations successfully from all the kernels we collected.

To examine kernel misconfigurations, we first manually reviewed the Android CDD and extracted the relevant kernel-level security mitigations. The CDD provides a set of mandatory kernel mitigations, which focus on addressing only a subset of security issues. However, these categories do not fully encompass the range of potential vulnerabilities that could compromise system security; therefore, we have examined prior works [64] and the additional mitigations that increase the security posture of the kernel. These mitigations and associated flags are present in Table 2. In total, we check 17 mitigations. The mitigations can be classified into three types - MUST (from CDD), STRONGLY RECOMMENDED (from CDD), and SUGGESTED (from [64]). Devices are expected to implement those which are MUST, and STRONGLY RECOMMENDED at least for a better security posture. Then, we have developed a

Attack Vector	Mitigation	Building Configuration
Stack Overflow	Stack Protector	** `CONFIG_HAVE_STACKPROTECTOR` or `CONFIG_STACKPROTECTOR{_STRONG}`
Control Flow Hijacking	KASLR	* `CONFIG_RANDOMIZE_BASE`
Heap Corruption	Freelist Randomization	`CONFIG_SLAB_FREELIST_RANDOM`
Information Leakage	USERCOPY	** `CONFIG_HARDENED_USERCOPY`
Buffer Overflow	Fortify Source	`CONFIG{_ARCH_HAS}_FORTIFY_SOURCE`
Code Injection	Non-executable Memory	** `CONFIG{_ARCH_HAS}_STRICT_KERNEL_RWX` or `CONFIG_DEBUG_RODATA`
Privilege Escalation	Restrict Userspace Memory Access	** `CONFIG_CPU_SW_DOMAIN_PAN` or `CONFIG_ARM64_SW_TTBR0_PAN`
Meltdown	Kernel Page Table Isolation	** `CONFIG_UNMAP_KERNEL_AT_EL0`
Spectre	Branch History Overwrite	`CONFIG_MITIGATE_SPECTRE_BRANCH_HISTORY`
Code reuse attacks	Enable CFI in kernel	* `CONFIG_CFI_CLANG` **and** `CONFIG_SHADOW_CALL_STACK`
Data Leaks	Prevent use of uninitialized local variables	+ `CONFIG_INIT_STACK_ALL{_ZERO}`
Data Leaks from Heap	Enable Heap Initialization	+ `CONFIG_INIT_ON_ALLOC_DEFAULT_ON`
Use After Free Vulnerability	Sanitization	`CONFIG_DEBUG_LIST`
Speculative Execution	BPF JIT enabled	`CONFIG_BPF_JIT_ALWAYS_ON`
Free list exploits	Harden Slab allocator	`CONFIG_SLAB_FREELIST_HARDENED`
Stack corruption	Virtually mapped kernel pages with guard pages	`CONFIG_VMAP_STACK`
Type confusion	Hardware enforced instructions	`CONFIG_ARM64_UAO`

Table 2: Attack vectors and mitigations in the kernel covered in our study; X{Y} means that either X or XY can be used for mitigation. ** means MUST; * means STRONGLY RECOMMENDED in Android 10; + means STRONGLY RECOMMENDED added in Android 12. For the configuration flags without any marks, they are the ones SUGGESTED by [64].
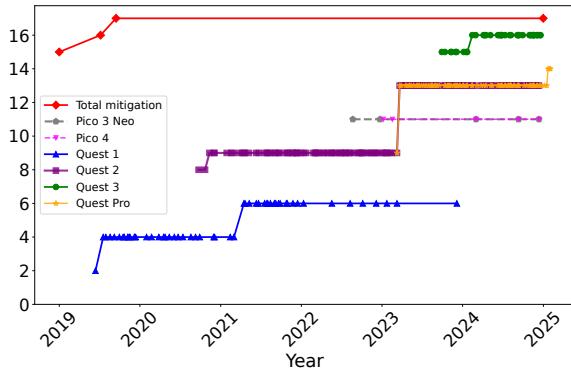
Fig. 3: Kernel mitigations adopted by the VR devices; for Pico devices we only have four versions of each.

script by taking the requirements as reference to analyze the kernel configuration (the extracted `.config` file) from the VR kernel binaries. We have examined all mandatory and suggested kernel-level mitigations to measure their adoption rates among VR devices.

4.3 Longitudinal analysis

In the VR firmware dataset we investigated, we observed that none of the firmwares adopted all the available mitigations. The mitigations adopted by different devices are shown in Fig. 3. Quest 1 began with only two kernel-level mitigations in mid-2019 and then was increased to six by early 2021, but it received no further enhancements despite additional mitigations becoming available. Quest 2 was launched in late 2020 with eight mitigations and reached nine within a few months, showing a similarly stagnant update pattern afterward. From the graph, we observe that the Quest Pro adopts the same kernel-level mitigations as Quest 2, and the updates were released on similar days. In contrast, newer models like Quest 3 adopted a significantly larger subset of available mitigations. The jumps in adopting kernel-level configurations in Quest, Quest 2 and Quest Pro are observed because of updates of Android versions, indicating that vendors would only include kernel-level mitigations during major Android version updates. In the case of the Pico 4

and Pico 3 Neo, we observed that the kernel version remains unchanged, along with the adoption of mitigations and there is no positive trend in adopting newer mitigations across the 4 versions we investigated.

Overall we find that Quest 3 has good security posture compared to other devices. On the positive side, more mandatory mitigations were adopted by Quest devices over time, even though they were not included in the initial versions. Although vendors may not always upgrade to the latest kernel versions, they add mitigations longitudinally through the device life cycle to enhance security over time. This approach, while improving security to some extent, does not fully mitigate the risks associated with outdated kernel versions. We also studied the missing kernel mitigations that were not adopted in the latest Quest and Pico devices in our dataset, shown in Table 4. We observe that both Quest and Pico devices miss many mitigations.

Kernel versions. One observation from our analysis is that VR devices tend to use older LTS (Long-Term Support) kernel versions than the latest LTS versions, the slower pace of Linux kernel adoption could explain the absence of certain kernel-level mitigations. The kernel version associated each device when they were released, along with the latest LTS version available, are shown in Table 3. We can see that the Quest, released in May 2019, adopted Linux kernel version 4.4, originally released in 2016, even though it could have adopted version 4.19, released in October 2018. Similarly, the Quest 2, launched in October 2020, continued to use Linux kernel version 4.19, despite the availability of version 5.4, released in 2019. Quest Pro was released in 2022, but it still adopted kernel version 4.19. The Quest 3, released in 2023, adopted Linux kernel version 5.10, even though version 6.1 had been available since 2022. All Pico headsets use the outdated Linux kernel version 4.19. Even the Pico 4, released in 2024, continued to rely on Linux 4.19, highlighting Pico's significant delay in adopting newer kernel versions. One possible reason vendors continue to use these older kernels is their focus on stability, as introducing newer kernels may cause issues that take time to resolve and require significant engineering efforts. However, this approach creates security gaps, as these devices are not adopting newer kernel-level mitigations.

Release date	Device	Kernel Version	Latest LTS	Android Version
May 2019	Quest	4.4.x	4.19.x	7.1.1
October 2020	Quest 2	4.19.x	5.4.x	10
October 2022	Quest Pro	4.19.x	5.15.x	10
October 2023	Quest 3	5.10.x	6.1.x	12
May 2021	Pico 3 Neo	4.19.x	5.10.x	10
October 2022	Pico 4	4.19.x	6.6.x	10

Table 3: Kernel versions used at the release time of each device and the latest LTS version available at that time.

Summary. VR devices demonstrate issues related to kernel misconfiguration. Meta Quest devices are striving to enhance the implementation of mitigations at the kernel level, while, in contrast, Pico devices show no evident longitudinal change in the adoption of kernel-level mitigations. Moreover, these devices are not proactively adopting the most recent stable Linux versions available. This delayed adoption exacerbates security vulnerabilities, as older kernels are deficient in incorporating newer mitigations vital for countering emerging threats.

5 RQ2: Binary Hardening

5.1 Motivation

Binary hardening [1] is a security technique that focuses on analyzing and modifying binary executable to safeguard against common exploits. This approach often involves non-deterministically altering control flow and instruction addresses, making it difficult for attackers to reuse program code and execute successful exploits. Binary compliance refers to whether a binary follows the hardening techniques prescribed by Android to prevent security issues. Lack of binary compliance can introduce vulnerabilities in the VR systems. This issue is demonstrated by incidents like the Stagefright vulnerability (CVE-2015-3864) [77] that exploited the multimedia framework binary that allows remote attackers to execute arbitrary code.

Although Android devices have increasingly adopted advanced binary hardening techniques [80], it remains unclear to what extent VR platforms, such as Meta Quest and Pico, implement these measures. In this section, our objective is to systematically investigate the adoption of binary hardening techniques on VR devices to identify potential inconsistencies and gaps. Inconsistent implementation may expose systems to vulnerabilities that might otherwise be avoided. For instance, Meta's application of Relocation Read-Only (RELRO) in Quest 2 [67] demonstrates how vital effective binary hardening is in ensuring the security of VR ecosystems.

5.2 Methodology

Our study focused on five specific mitigation techniques that should always be enabled according to CDD: Stack Canaries, Full Relocation Read-Only (Full RELRO), Fortify Source, Control Flow Integrity (CFI), and the NX bit, These techniques collectively work to defend against prevalent exploits and minimize potential vulnerabilities. Our approach to identifying these mitigations was adapted from methodologies outlined in past research [80, 105], which primarily explored these mitigations within embedded systems and Android environments, which we detail below.

- **Stack Canaries** are a defense mechanism against stack-based buffer overflow vulnerabilities by detecting any stack corruption before the function returns. To verify the implementation of Stack Canaries, we check `__stack_chk_fail` within symbols or the Global Offset Table (GOT) table.

- **Control Flow Integrity (CFI)** protects against attacks such as Return-Oriented Programming (ROP) and Jump-Oriented Programming (JOP) by maintaining the correct control flow. We verify the binary's symbol table for the existence of `.cfi` symbols.

- **Fortify Source** aims to improve memory safety by identifying potentially unsafe uses of standard library functions during the compilation process. Detection involves searching for `*_chk` functions within the binary; additional indicators include strings like `*buffer overflow detected*`, which signify binary fortification.

- **NX bit** is a defense against buffer overflow attacks by designating certain memory regions as non-executable, thereby preventing code execution in those areas. We inspected the binary for the `PT_GNU_STACK` segment. If found and marked non-executable, it signifies that the binary seeks a Non-executable Stack.

- **Full RELRO** protects critical relocation sections of a binary by rendering them read-only, thereby preventing GOT overwrite attacks. The absence of a `GNU_RELRO` segment indicates that the binary lacks RELRO protection.

5.3 Longitudinal analysis

Binaries without Canaries, CFI, and Fortify Source. The trend of binaries without canaries is shown in Fig. 4. Quest 1 had around 150 system binaries without canaries in 2019, gradually increases to about 300 by 2023, and then remains flat. Quest 2 enters the market in late 2020 with around 350 binaries without canaries, stays relatively stable at about 300 through 2022-2023, then increases rapidly in 2023-2024. Quest 3 and Quest Pro both show similar patterns (overlapping). All headsets except Quest 1 show a dramatic spike reaching peaks of around 1000 binaries without canaries in early 2024. After this spike, there's a significant drop, followed by a period of stabilization through 2025. The Quest 1 appears to be the only Quest headset that did not experience the dramatic spike in late 2023. Pico 4 did not experience any noticeable changes, while Pico 3 Neo had an increase in mid 2024. The CFI (Fig. 5) and Fortify Source (Fig. 6) show a similar trend, albeit the detailed numbers are different.

Binaries without NX. As observed in Fig. 7, Quest 1 maintains a consistent level of binaries without NX until the start of 2021, after which there is a notable drop, then it fluctuates through 2024. This suggests a significant enhancement for enforcing NX in 2021, coinciding with an Android version update. Since 2021, Quest 2's count of binaries without NX remained relatively stable but saw a spike in 2023 due to an Android upgrade, followed by fluctuations into 2024. Quest Pro exhibits a trend similar to Quest 2, though with slightly elevated values at some points in 2023. Quest 3 begins with strong NX protection (indicated by a low number of binaries without NX), and by 2024, this count is nearly halved. Pico 4 does not show any change, while Pico 3 Neo has a significant jump of binaries without NX in mid 2024.

Binaries without RELRO. As observed in Fig. 8, Quest 1 starts extremely high at around 280 binaries with no RELRO protection in early 2019, then shows a dramatic decline over 2019, stabilizing

Device	Kernel Version	Missing Kernel Mitigations
Quest	4.4.x	*CONFIG_CFI_CLANG, CONFIG_SHADOW_CALL_STACK, CONFIG_SLAB_FREELIST_RANDOM, CONFIG_FORTIFY_SOURCE, CONFIG_MITIGATE_SPECTRE_BRANCH_HISTORY, CONFIG_DEBUG_LIST, CONFIG_BPF_JIT_ALWAYS_ON, CONFIG_SLAB_FREELIST_HARDENED, CONFIG_VMAP_STACK, CONFIG_ARM64_UAO
Quest 2 / Quest Pro	4.19.x	**CONFIG_UNMAP_KERNEL_AT_EL0, + CONFIG_INIT_STACK_ALL_ZERO, +CONFIG_INIT_ON_ALLOC_DEFAULT_ON, *CONFIG_CFI_CLANG
Quest 3	5.10.x	+CONFIG_INIT_ON_ALLOC_DEFAULT_ON
Pico 3 / Pico 4	4.19.x	**CONFIG_ARM64_SW_TTBR0_PAN, **CONFIG_UNMAP_KERNEL_AT_EL0, CONFIG_MITIGATE_SPECTRE_BRANCH_HISTORY, *CONFIG_CFI_CLANG, CONFIG_DEBUG_LIST, CONFIG_BPF_JIT_ALWAYS_ON

Table 4: Missing kernel mitigations in latest firmware versions.
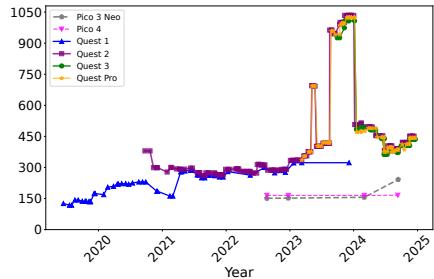
Fig. 4: Binaries without Canaries
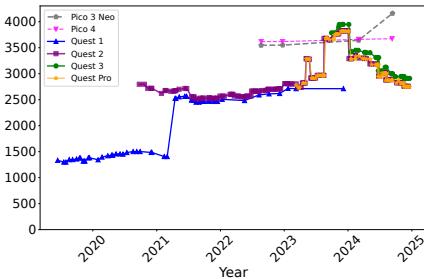
Fig. 5: Binaries without CFI
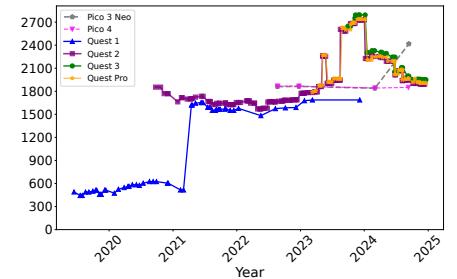
Fig. 6: Binaries without Fortify Source
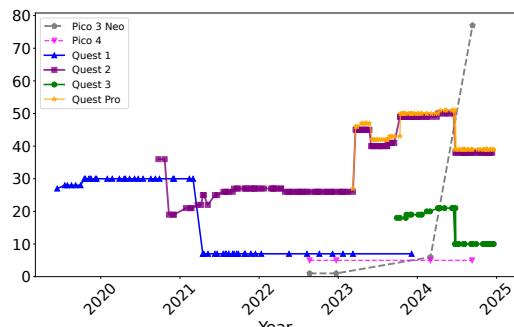
Fig. 7: Binaries without NX
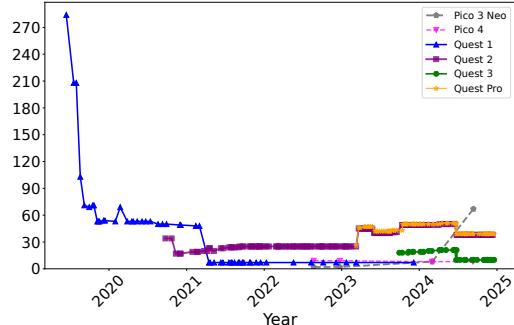
Fig. 8: Binaries without RELRO

around 50-60 through 2020. In early 2021, it drops sharply again to near zero and remains at that minimal level through 2023. Quest 2 starts in late 2020 with a count much lower than Quest 1's initial values. It maintains this level through 2022-2023, then fluctuates but generally maintaining this range through 2024. Quest Pro shows the same trend as Quest 2. Quest 3 starts in late 2023 with the lowest initial values of any headset (around 20), suggesting it launched

with better RELRO protection from the start, and then drops to even lower levels in 2024. This shows Meta's progressive approach in enhancing the RELRO protection in Quest devices. Similar to other results of Pico devices, Pico 4 does not show any change, while Pico 3 Neo has a significant jump in mid 2024.

Spikes. We observed noticeable spikes in the number of not hardened binaries for Quest 2 and Quest Pro after 2023 (Fig. 4, Fig. 5, Fig. 6). The increase was due to the addition of binaries in preinstalled apps, most of which lacked hardening. All the added binaries had names starting with "libxplat". In subsequent updates, many of these binaries were removed, specifically the same ones that had been previously added. Upon investigation, we found that these binaries were intended to provide cross-platform functionality. We conjecture that vendors might have opted out of these hardening techniques due to performance reasons [20], as using these hardening techniques may penalize performance.

Latest version. We analyze the latest versions in Appendix A of [85].

> **Summary.** Within the VR ecosystem, vendors frequently neglect binary hardening techniques, particularly during major Android version upgrades. While longitudinally we see a slow but steady reduction in binaries lacking such security measures, the consistent absence of CFI and Fortify Source across thousands of binaries reveals a significant gap in binary hardening in VR devices.

6 RQ3: Preinstalled Apps

6.1 Motivation

Preinstalled apps are those that come with the firmware; the user cannot uninstall these apps without root. Previous studies [32, 33, 45] have identified the presence of security or privacy-related issues in pre-installed apps, such as privilege escalation, command

execution, etc. For example, CVE-2018-9525 [21] demonstrated privilege escalation due to permission bypass; a broadcast receiver did not enforce `android.permission.MANAGE_SLICE_PERMISSIONS`, which caused privilege escalation. In this section, we first study the ecosystem of the VR system apps, analyze app manifests, and check how the permission usage and custom permission declaration change across the firmware versions in the preinstalled apps.

6.2 Methodology

We study two aspects of preinstalled apps on VR devices: security flags in manifest files, and permissions.

6.2.1 Security flags in manifest files. The `AndroidManifest.xml` file is like a blueprint for an app, it declares essential information such as the app's package name, components, required permissions, hardware features, and the minimum Android version it supports. The system uses it to understand how to launch and run the app. For investigating the manifest of system apps, we use Androguard [23] to decompile the `apk` file and develop a custom script to check the presence of security flags in the manifest of the apk file. We compile the following list of security flags based on prior work [45]:

- `allow_backup` specifies whether the app's data can be backed up and restored via Android's backup mechanism. Setting the `allow_backup` flag to `true` allows user data to be stored in the cloud. However, this may result in the unintended exposure of sensitive information, particularly if the application stores sensitive user or private data.

- `debuggable` indicates whether an application can be debugged. Typically, this flag is set to true in debug builds and false in release builds. If this flag is set to `true` in a release build, attackers can leverage debugging tools to inspect the internal state of the application, examine variable values, and potentially uncover sensitive information or vulnerabilities.

- `use_cleartext_traffic` indicates whether an application allows plaintext (unencrypted) HTTP traffic. Setting this flag to `true` allows the application to use plain-text HTTP connections. This makes the app susceptible to man-in-the-middle (MITM) attacks, as unencrypted data transmitted over the network can be intercepted and altered.

6.2.2 Permission analysis. We have developed a custom script based on Androguard [23] to extract all the permissions used along with their protection levels from the framework-level apk files (`com.oculus.os.platform-res.apk`, `horizonos.platform-res.apk` and `framework-res.apk`, which define system-wide permissions) as well as preinstalled apps (which define custom permissions). We further study three aspects of permission issues: 1) permission categorization based on app types; 2) change of permission protection level; and 3) permission inconsistencies.

Permission categorization based on app types. First, we classify the apps into three categories. **User-launchable apps** have a user interface that can be launched by users. We parsed the manifest file to check the presence of `android.intent.action.MAIN` and `android.intent.category.LAUNCHER`. **Android system apps** are part of the Android system. We identified these apps by examining package names that follow the format `com.android.*`. **Vendor specific apps** are the remaining apps, including the vendor-specific

Device	use_cleartext_traffic	allow_backup	debuggable
Quest	6	0	0
Quest 2	9	1	0
Quest Pro	9	1	0
Quest 3	9	1	0
Pico Neo 3	27	8	0
Pico 4	27	7	0

Table 5: Security flags in the latest version of VR devices.

ones (e.g., `com.oculus.*`). Together, these three categories encompass all preinstalled applications.

Change of permission protection levels. Once the applications and their respective protection levels were identified, we marked the application permissions that displayed varying protection levels across device firmware updates. This suggests that the privileges associated with specific permissions were either enhanced or reduced for certain reasons. For instance, if a permission initially set to `normal` is later elevated to `system`, this suggests that there may have been an event necessitating tighter control.

Permission inconsistencies. We have further analyzed two kinds of inconsistent permission scenarios: *Residual Permission:* permissions that are declared in the system but not used by any preinstalled apps; and *Phantom Permission:* permissions that are used by preinstalled apps but not declared in the system. In both instances, the permission settings are improperly configured.

6.3 Longitudinal Analysis

6.3.1 Security flags in manifest files. Table 5 displays the current security flags present in the manifest files of the newest VR device models. Meta Quest devices have a low number of apps with the `use_cleartext_traffic` flag set, reflecting a stricter stance against allowing unencrypted traffic. Conversely, Pico devices have a much higher number of apps with `use_cleartext_traffic` enabled, implying a more lenient approach to cleartext traffic that suggests potential security risks. For `allow_backup`, the original Quest device does not permit backups; however, Quest 2, 3, and Pro models provide for limited backup capabilities. In contrast, Pico devices have a much larger number of preinstalled apps allowing backup, indicating more liberal backup policies, which could pose privacy risks. Regarding the `debuggable` flag, these are universally disabled across all devices, ensuring that debugging is turned off in production builds.

6.3.2 Permission analysis. We present the permission statistics in terms of different protection levels of the latest firmware versions in Table 6. For the latest version, for Quest and Pico devices, the mean number of `dangerous`, and `normal` permissions requested per app was similar. However, Quest devices request a significantly larger number of `others` permissions (2-3x) compared to Pico devices. This is due to the addition of a large number of custom permissions (*e.g.*, 331 in Quest 3) to the Quest device family; Pico does not add these many custom permissions (*e.g.*, 63 in Pico 4). Moreover, the VR devices have introduced 17 new `dangerous` permissions, including `FACE_TRACKING`, `BODY_TRACKING`, and `EYE_TRACKING`, which can be requested by userspace applications.

Permission changes. For Pico devices, we did not observe many permission changes in their preinstalled apps, so we focus on Quest devices in the remainder of this section. The longitudinal permission

Device	Dangerous	Normal	Signature	SigOrSys	Others
Quest	2.02	2.93	2.77	0.7	4.87
Quest 2	2.63	3.67	4.09	0.75	8.30
Quest Pro	2.62	3.68	4.08	0.74	8.56
Quest 3	2.64	3.64	4.04	0.74	8.17
Pico Neo 3	1.95	2.92	1.33	0.62	2.86
Pico 4	1.92	2.80	1.29	0.61	2.87

Table 6: Mean of permission counts of the latest firmware; SigOrSys: SignatureOrSystem; Others: all other permissions.

Fig. 9: Permission Changes of Quest 2

Fig. 10: App Categorization of Quest2

Fig. 11: Phantom and Residual Permissions of Quest 2

which are selected from a total of 64 apps that are present across all firmware versions of Quest 2. Several apps consistently show very high permission counts across all firmware versions, such as SystemUX.apk, OCMS.apk and VrShell.apk.

Change of permission protection levels. In early versions of the Oculus Quest, we found that critical Android permissions such as REQUEST_INSTALL_PACKAGES and SET_TIME_ZONE were incorrectly marked as normal. Permissions with a normal protection level can be requested by any app without user approval, potentially allowing malicious apps to disrupt system behavior. With permissions like SET_TIME_ZONE, third-party apps can disrupt logs, and with REQUEST_INSTALL_PACKAGES permission, third-party apps can get the privilege to install other apps. In later versions, the permission protection level was changed to signature. Additionally, we identified that a custom permission RECORD_AUDIO_BACKGROUND was initially assigned a dangerous protection level in Quest and Quest 2. Dangerous permissions can be granted to third-party apps with user consent, creating a privacy risk, especially for a permission that allows background audio recording. In later versions (Quest 3 and Quest Pro), this permission was completely removed, probably due to its sensitive nature.

Another concerning trend was the change in protection levels for permissions like FACE_EYE_INTERNAL_API and ORTHOFIT_DATA. These were originally marked as preinstalled in Quest and Quest 2 — meaning only system apps could request them — but were later changed to dangerous. This shift allows third-party apps to request these sensitive permissions, increasing the potential for misuse.

Permission inconsistencies. We present the number of phantom and residual permission counts for preinstalled apps in Quest 2 over time in Fig. 11. We can see that the phantom permissions increased over time (from 20 to over 80), but residual permissions fluctuated (between 250 to 300). An instance of residual permission is PERFORM_SIM_ACTIVATION; although VR devices lack support for SIM cards, this permission is still specified at the framework level across all Quest device families. A phantom permission example is CAR_CONTROL_AUDIO_VOLUME, a system-level permission for adjusting the car's audio volume when connected via Android Auto. This permission was applied in the Quest 2 within the VRSystemUI.apk app without being declared, and it was later removed through updates. Another interesting phantom permission case is com.oculus.permission.HAND_TRACKING; this permission was used by preinstalled apps even before it was declared in the framework-level apk files, which may be the cause of some

changes for Quest 2 are shown in Fig. 9 (similar figures for other devices are presented in Appendix A of [85]). We also plot the number of apps in the same figure. We find that the number of dangerous permissions have remained relatively stable across firmware versions, while normal, signature, and SignatureOrSystem permissions show steady growth over time. others permissions experienced a significant rise, escalating from approximately 300 to over 1000. This suggests a substantial number of custom permissions were introduced. The most widely adopted permission longitudinally is READ_PRIVILEGED_PHONE_STATE. This permission is used to obtain the device identifiers. Other commonly requested permissions included ALWAYS_CAPTURE_MIC_AUDIO_INPUT, RECORD_AUDIO, and START_ACTIVITIES_FROM_BACKGROUND.

Permission categorization based on app types. The app categorization of Quest 2 is shown in Fig. 10. Android system apps showed no significant increase in permission usage except during Android OS upgrades, while User launchable and Vendor specific apps showed significant increases across versions. Vendor apps represent more than 50% of the apps. In Fig. 12, we present top-20 Vendor apps with highest permission numbers in Quest 2 (similar figures for other devices are presented in Appendix B of [85]),

Fig. 12: Top-20 Vendor apps with highest permission numbers in Quest 2.

functionality bugs [96]. Later, the permission was declared in the framework-level apk files.

Summary. VR devices are prone to vulnerabilities such as accidental data backups and exposure to MITM attacks due to vulnerable security flag configurations. Problems like permission inconsistencies, residual permissions, and phantom permissions highlight the security concerns in permission management.

7 RQ4: SEPolicy

7.1 Motivation

SEPolicy is a set of rules used by SELinux to enforce Mandatory Access Control (MAC) within a system. It specifies how processes (subjects) can interact with other processes and system resources (objects). Vendors often customize the default Android SEPolicy to suit their devices. However, incorrect or overly permissive modifications can introduce security vulnerabilities. By default, SELinux adopts a *deny* approach, meaning any action not explicitly permitted by a rule is denied. Analyzing SEPolicy for VR devices is crucial because these devices often include customized system components, sensors, and privileged services that aren't present in standard Android environments. These additions may require vendor-specific policy changes, which, if misconfigured, could unintentionally weaken security boundaries—especially given the sensitive data and capabilities (e.g., camera, motion tracking) involved in VR platforms.

7.2 Methodology

In the firmware images, the SEPolicy files (`.cil` files) are present in `/etc/selinux` in `system.img` and `/etc/selinux` in `vendor.img`. After extracting the `.cil` files, we have processed them to extract the `allow` (explicitly permit specific actions) and `never-allow` rules (prevent certain permissions from being granted). As early Quest 1 firmware (running Android 7) only contains SEPolicy binaries instead of `.cil` files, we exclude them from our study; we only include 24 Quest 1 firmware ranging from April 2021 to December 2023 (running Android 10).

7.3 Longitudinal Analysis

Fig. 13 shows the evolution of SEPolicy rules across different VR devices. Longitudinally, the number of `never-allow` rules remains stable for all devices. The only increase appeared in 2023 for Quest 2, where it upgraded from Android 10 to 12. For `allow` rules, the numbers grow gradually for Quest and Pico devices. There are many more `allow` rules than `never-allow` rules for all devices.
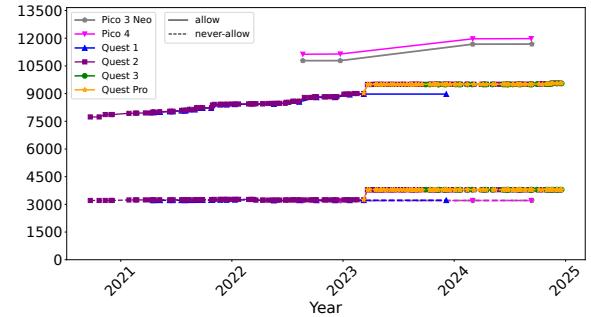
Fig. 13: SEPolicy: Allow and Never-allow Rules

Isolated and untrusted app domains. Isolated and untrusted app domains in SEPolicy are intended to regulate access by third-party applications. Through our analysis of `allow` and `never-allow` rules targeting these domains, we observed notable differences between Quest and Pico devices. Specifically, all devices within the Quest family retained the default Android SEPolicy configurations, showing no customizations. In contrast, Pico devices exhibit significant changes to these policies. Pico has explicitly removed certain `never-allow` rules associated with isolated and untrusted app domains. The removal of such rules weakens the SEPolicy's rigor on Pico devices. For example, in Pico devices, the never-allow rules corresponding to `net_dns_prop`, `radio_cdma_ecm_prop` were removed. Moreover, Pico allows untrusted applications to access vendor-specific property files such as `vendor_android_pvr_prop`, which is a property file introduced by Pico. Additionally, another property, `hwservicemanager_prop`, typically restricted by Android, is accessible to untrusted apps on Pico. These property files usually contain unique device identifiers, and their exposure poses privacy risks such as device fingerprinting [25, 65].

Summary. VR devices often customize SEPolicies. We found that some `never-allow` rules have been removed in VR devices, which may lead to security issues. Additionally, applications from untrusted domains can access property files, potentially leading to privacy issues like device fingerprinting.

8 Discussion

Our longitudinal study of VR firmware reveals several recurring and systemic security weaknesses across kernel configuration, binary hardening, application privilege enforcement, and SELinux policy design. While some protections are partially adopted, the

overall security posture of VR firmware lags behind modern Android standards. Below, we offer concrete recommendations for VR device vendors and discuss the limitations of our study.

8.1 Impacts of Our Findings

Lack of proper kernel mitigations (RQ1) can allow privilege escalation attacks that tamper with raw sensor data (e.g., eye-tracking) or camera feeds. Similarly, weak hardening (RQ2) and mismanaged permissions (RQ3) can let third-party or preinstalled apps exploit privileged system services, posing risks in sensitive VR applications such as medical or military training. In addition, incorrect or overly permissive modifications of SEPolicy (RQ4) can introduce security vulnerabilities to the VR platforms, leading to privacy violations (*e.g.*, leakage of sensitive motion data) or privilege escalations.

We found that VR-specific permissions such as `ORTHOFIT_DATA` and `FACE_EYE_INTERNAL_API`, used for tracking, were changed from `preinstalled` to `dangerous`, exposing sensitive data to third-party apps and reflecting emerging attack surfaces not present in traditional Android systems. In addition, SEPolicy customizations in VR firmware can enable untrusted apps to access vendor-defined properties related to motion tracking. We also observed that binaries introduced in recent VR firmware (e.g., `libsensorutils`, `pvr_compute`) often lacked standard hardening protections, suggesting a trade-off between real-time responsiveness and security, that is specific to VR.

8.2 Responses from Vendors

Following responsible disclosure practices, we reported our findings to Meta and Pico, who have acknowledged our findings. While both vendors requested proof-of-concept exploits to validate the security implications, the closed nature of these VR platforms precludes dynamic analysis capabilities, as detailed in §8.4. Consequently, our empirical study identifies potential security and privacy vulnerabilities through static analysis, rather than demonstrating active exploits. These findings underscore systemic weaknesses in VR firmware security that merit further investigation.

Pico's response. Through iterative communications with Pico's development team, we received comprehensive technical responses regarding our identified security concerns. Pico, like other OEM vendors, inherits kernel source code directly from SoC manufacturers such as Qualcomm. The kernel version is intrinsically coupled with the underlying SoC hardware, making kernel upgrades infeasible without corresponding hardware changes. This architectural constraint explains why Pico Neo 3 and Pico 4, both utilizing Qualcomm XR2 SoC, maintain the same 4.19 kernel version. The development team has committed to maintaining security through continuous integration of upstream vulnerability patches.

The vendor noted that binary hardening mechanisms (Stack Canaries, CFI, Fortify, NX, RELRO) alone do not determine program security, as runtime context and program functionality must also be considered. Their security policy mandates hardening only for high-risk binaries that expose external services or have known vulnerabilities. The vendor cited technical challenges in implementing universal hardening across their large codebase, including legacy code compatibility issues and high engineering costs associated with modifying existing binaries.

Regarding `use_cleartext_traffic` flag, Pico explained that despite the flag being enabled, their proprietary network library enforces HTTPS, ensuring encrypted network traffic. They also noted that `allow_backup` is an intended Android OS feature rather than a vulnerability, though we analyzed it following prior work [45]. Pico provides SDKs that enable third-party developers to access device features and enable SEPolicy rules. Their security team is actively reviewing and refining SEPolicy configurations to eliminate unnecessary rule modifications in future releases.

Meta's response. Meta provided targeted feedback on specific findings. They clarified that applications with `use_cleartext_traffic` enabled are inherited from AOSP defaults and largely inactive in HorizonOS. For sensitive permissions like `ORTHOFIT_DATA` and `FACE_EYE_INTERNAL_API`, Meta implemented system-level access controls restricting usage to authorized system applications. Their permission deployment strategy involves initial testing with first-party applications before broader third-party availability, which may explain the observed permission level transitions.

8.3 Recommendations for Stakeholders

For vendors. A consistent theme across all VR firmware we analyzed is the partial and inconsistent adoption of core Android security mechanisms. This is mainly because VR firmware development currently lacks a standardized, enforceable baseline like Android's Compatibility Definition Document (CDD). As a result, vendors vary significantly in their implementation of core security practices. We recommend the VR vendors adopt a formal compliance framework that specifies minimum requirements, such as kernel configurations, compiler flags, SELinux policies, and permission enforcement rules, modeled on the Android CDD but specific to VR. Strict SEPolicy isolation should be mandated for new VR system services, with least-privilege boundaries clearly enforced.

The limited use of advanced mitigations in VR may reflect perceived tradeoffs between performance and security, especially in latency-sensitive VR environments. Prior works [64, 84, 108] have demonstrated that hardening techniques, such as CFI, full RELRO, or updated kernel configurations, can introduce overhead that may affect rendering performance or real-time sensor processing. However, our findings suggest that vendors are making these tradeoffs without fully accounting for the security implications. We recommend that in cases where vendors disable mitigations due to performance constraints, these decisions should be transparently documented and justified. Alternative defenses, such as stronger sandboxing, isolation, or monitoring, should be employed to compensate. Performance-aware variants of mitigations (e.g., lightweight CFI) should also be explored. For instance, Site Isolation in Chrome [81] demonstrates how performance and security can be jointly optimized in large-scale platforms.

Our analysis of preinstalled apps and SELinux policies reveals that configurations often remove `neverallow` rules, introduce unused domains (e.g., telephony services on non-cellular VR devices), or fail to isolate new capabilities. Preinstalled apps frequently request excessive permissions, including access to sensitive data (e.g., `FACE_TRACKING` and `EYE_TRACKING`), with protection levels changing

across updates. These patterns suggest inadequate review and enforcement. We recommend that vendors regularly audit app permissions and SELinux policies, apply strict protection levels to custom permissions, and isolate all new services using fine-grained SELinux domains. Legacy and unused policies should be aggressively pruned to reduce the attack surface and improve policy clarity.

For developers. VR applications are often built using complex game engines and development frameworks (e.g., Unity, Unreal), which offer rapid development workflows but may enable insecure defaults. Our analysis shows that many apps depend on untrusted third-party components for core functionality such as motion tracking and input handling. We recommend that developers disable insecure defaults (e.g., `allowBackup=true`, overly broad intent filters) and avoid exposing sensitive components unnecessarily. When using Unity or Unreal, build configurations should be explicitly hardened following platform guidance (e.g., through ProGuard, and code obfuscation). In addition, developers should avoid incorporating third-party libraries for privacy-sensitive functionality (e.g., eye tracking, facial capture, and spatial tracking) unless they are thoroughly vetted. Developers should request only those permissions essential to app functionality and avoid over-scoping permission declarations, especially for biometric or sensor data.

For manufacturers. Many modern VR headsets are built on top of mobile SoCs that include hardware-backed security features (e.g., pointer authentication, TEEs). However, we found that these capabilities are often not enabled or not fully integrated into production firmware. We recommend that chipset manufacturers collaborate closely with headset manufacturers to ensure that hardware capabilities are fully leveraged and enabled by default in VR firmware builds. This includes providing reference firmware configurations, performance benchmarks, and integration guidelines to reduce adoption barriers.

8.4 Limitations

No source code available for VR devices. VR devices run on tailored Android systems, with modifications being much more extensive than those seen in standard Android smartphones. Despite this, no VR device manufacturer has publicly released its source code. This makes it extremely challenging, if not impossible, to infer the underlying reasons for our findings observed in firmware. For instance, consider the situation involving residual permissions; had the source code been accessible to us, we could have conducted a more thorough analysis regarding the presence of these residual permissions and their potential implications. This limitation also restricts our ability to study customizations in the framework layer.

No dynamic analysis. Currently, gaining root access on commercial VR devices is not feasible (Except Oculus Go [69]). As a result, we are unable to escalate privileges or gain system-level access required to perform dynamic analysis, which is achievable in prior Android firmware papers [56]. In addition, major VR vendors do not i) provide official simulators, or ii) expose low-level APIs or debugging endpoints for fuzzing or instrumentation. This limitation restricts our ability to monitor runtime behaviors, detect dynamic privilege use, or uncover real-time data flows and interprocess communication. Consequently, our analysis is constrained to a static approach.

Limited VR firmware datasets. In our study, we focused primarily on the VR firmware of the Meta Quest Series and Pico. Although there are additional VR devices available, such as the HTC Vive and Varjo, we were unable to locate any available resources regarding their firmware. Due to the lack of root access, extracting firmware images directly from the devices is highly challenging. A possible solution involves capturing the over-the-air updates provided to these VR devices, which include the updated firmware images. We leave a thorough exploration of such methods as our future work.

Generalizability. While our research questions and findings are designed for Android-based VR platforms, and can extend to future Android-XR devices, they may not apply to headsets running fundamentally different OS (e.g., Apple's visionOS).

9 Related Work

Android firmware analysis. Existing works have performed analysis on Android firmware to study security issues. Several works carry out measurement studies on hundreds of Android firmware from different smartphone vendors [33, 47, 48]. Researchers have also proposed different firmware analysis tools to study different aspects such as pre-installed apps [32, 92], security policies [46], AT commands [93], init routines [52] and unix domain sockets [30].

Android customization. Another important aspect of Android is the customization introduced by device vendors, as demonstrated by existing works [4, 28, 53, 60, 80, 100, 104, 113]. Third-party libraries are an important component of the Android ecosystem. Many existing works aim to identify third-party libraries and their vulnerabilities in Android apps [15, 22, 58, 99, 101, 106, 110]. In our work, we focus on analyzing firmware of VR systems.

VR application analysis. Other researchers have assessed VR apps or SDKs to explore privacy concerns. In OVRSeen [95], the authors conducted an investigation into privacy breaches in Oculus applications by analyzing network traffic. Subsequently, a tool named VPVet [107] was developed to evaluate the privacy policies of VR applications. A recent study [45] has examined third-party VR apps. Different from these works, our research targets a longitudinal analysis of VR firmware.

VR attacks. Previous studies have revealed that privacy leakages in VR environments can be exploited for user deanonymization attacks. Most of these studies utilize built-in sensors (e.g., accelerometer and gyroscope) in VR headsets and analyze user motion data [14, 50, 55, 59, 72, 73, 78, 79, 87, 94, 111], whereas a few of them employ specialized external devices to explore other user data [49, 83], e.g., electroencephalogram. The attacker may also utilize side channels to obtain sensitive information. Secrets such as keystrokes in VR can also be inferred from motion data [62, 71, 87, 111] and other side-channel information, such as videos of user movements [44, 76, 103], acoustic signals [63], network traffic [91] and WiFi signals [5]. Recent works [57, 74, 75] have proposed injecting noise into motion data or identifiable anthropometrics to make inference attacks harder, which have predominantly used differential privacy [26, 27]. However, these works did not study the firmware of VR systems.

10 Conclusion

In this paper, we conduct the *first* in-depth security analysis of more than 300 VR firmware over time from two major vendors, Quest and Pico. Our study uncovers critical security issues within VR firmware, including absent kernel and binary protections, inconsistent permission settings, and improper removal of SELinux policy enforcements. This research sheds light on the largely uncharted area of VR firmware security, providing valuable insights that can guide the development of future VR systems.

Acknowledgement

References

[1] "Hardening (computing) - Wikipedia — en.wikipedia.org," https://en.wikipedia.org/wiki/Hardening_(computing).

[2] "Htc community forum," https://forum.htc.com/.

[3] "Ikconfig - linux kernel configuration database (lkddb)," https://cateee.net/lkddb/web-lkddb/IKCONFIG.html.

[4] Y. Aafer, X. Zhang, and W. Du, "Harvesting inconsistent security configurations in custom android {ROMs} via differential analysis," in *USENIX Security*, 2016.

[5] A. Al Arafat, Z. Guo, and A. Awad, "Vr-spy: A side-channel attack on virtual key-logging in vr headsets," in *IEEE VR*. IEEE, 2021, pp. 564–572.

[6] J. Alakuijala and Z. Szabadka, "Brotli compressed data format," RFC 7932. [Online]. Available: https://www.rfc-editor.org/info/rfc7932

[7] Android Developers, "Permissions on android," https://developer.android.com/guide/topics/permissions/overview.

[8] ——, "Use of native code," 2025. [Online]. Available: https://developer.android.com/privacy-and-security/risks/use-of-native-code

[9] Android Open Source Project, "Architecture overview," 2025. [Online]. Available: https://source.android.com/docs/core/architecture

[10] ——, "Compatibility definition document," 2025. [Online]. Available: https://source.android.com/docs/compatibility/cdd

[11] Apple, "Introducing apple vision pro," https://www.apple.com/apple-vision-pro/.

[12] AR Insider, "Consumer and enterprise virtual reality (vr) market revenue worldwide from 2021 to 2026 (in billion u.s. dollars)," 2022. [Online]. Available: https://www.statista.com/statistics/1221522/virtual-reality-market-size-worldwide/

[13] I. Archive, "Pico os software," 2025. [Online]. Available: https://web.archive.org/web/20250000000000*/https://www.picoxr.com/global/software/pico-os

[14] S. Aziz and O. Komogortsev, "Exploring the uncoordinated privacy protections of eye tracking and vr motion data for unauthorized user identification," in *IEEE VR*. IEEE, 2025, pp. 217–227.

[15] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *CCS*, 2016.

[16] 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.

[17] V. Corporation, "Valve index headset technical specifications," 2019. [Online]. Available: https://www.valvesoftware.com/en/index/headset

[18] Counterpoint Research, "Global XR (AR & VR Headsets) Market Share: Quarterly," https://www.counterpointresearch.com/insight/global-xr-ar-vr-headsets-market-share-quarterly, 2025.

[19] CVE, "Cve," https://nvd.nist.gov/vuln/detail/cve-2018-9568.

[20] T. H. H. Dang, P. Maniatis, and D. Wagner, "The performance cost of shadow stacks and stack canaries," in *AsiaCCS*, 2015, pp. 555–566.

[21] N. V. Database, "CVE-2018-9525 Detail," https://nvd.nist.gov/vuln/detail/CVE-2018-9525, 2018.

[22] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on android," in *CCS*, 2017.

[23] A. Desnos *et al.*, "Androguard: Reverse engineering, malware and goodware analysis of android applications," https://github.com/androguard/androguard, 2011–2024.

[24] X. Developers, "Xda developers forums," https://xdaforums.com/.

[25] Z. Dong, L. Wang, G. Xu, and H. Wang, "On the (in)security of non-resettable device identifiers in custom android systems," *arXiv preprint arXiv:2502.15270*, 2025.

[26] C. Dwork, "Differential Privacy," in *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4052, pp. 1–12, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/11787006_1

[27] ——, "Differential Privacy: A Survey of Results," in *Theory and Applications of Models of Computation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 4978, pp. 1–19, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-540-79228-4_1

[28] Z. El-Rewini and Y. Aafer, "Dissecting residual apis in custom android roms," in *CCS*, 2021.

[29] S. Electronics, *Samsung Gear VR Operating Instructions*, 2015. [Online]. Available: https://www.bhphotovideo.com/lit_files/132118.pdf

[30] M. Elgharabawy, B. Kojusner, M. Mannan, K. R. Butler, B. Williams, and A. Youssef, "Sausage: security analysis of unix domain socket usage in android," in *(EuroS&P)*. IEEE, 2022.

[31] ellie, "Vr firmware download - quest series," https://cocaine.trade/.

[32] M. Elsabagh, R. Johnson, A. Stavrou, C. Zuo, Q. Zhao, and Z. Lin, "{FIRMSCOPE}: Automatic uncovering of {Privilege-Escalation} vulnerabilities in {Pre-Installed} apps in android firmware," in *USENIX Security*, 2020.

[33] J. Gamba, M. Rashed, A. Razaghpanah, J. Tapiador, and N. Vallina-Rodriguez, "An analysis of pre-installed android software," in *S&P (Oakland)*. IEEE, 2020.

[34] gameishard, "Virtual reality use increases concerns for child safety, study shows," https://gameishard.gg/news/over-75-believe-kids-at-major-risk-of-sexual-abuse-when-using-vr-headsets-report/92677/.

[35] E. Games, "Unreal engine forums," https://forums.unrealengine.com/.

[36] Github, "Boot-img-extractor," https://github.com/cfig/Android_boot_image_editor.

[37] ——, "Extract-ikconfig," https://github.com/torvalds/linux/blob/master/scripts/extract-ikconfig.

[38] ——, "Github," https://github.com/QuestEscape/exploit/tree/master/CVE-2018-9568_WrongZone.

[39] Google, "Android security bulletins," https://source.android.com/docs/security/bulletin/asb-overview.

[40] ——, "Specifications for viewer design - cardboard manufacturer help," 2015. [Online]. Available: https://support.google.com/cardboard/manufacturers/answer/6323398?hl=en

[41] ——, "Hardening the kernel in android oreo," https://android-developers.googleblog.com/2017/08/hardening-kernel-in-android-oreo.html, 2017.

[42] ——, "Android xr: The gemini era comes to headsets and glasses," 2025. [Online]. Available: https://blog.google/products/android/android-xr/

[43] ——, "The compatibility test suite (cts) overview," https://source.android.com/docs/compatibility/cts, 2025.

[44] S. R. K. Gopal, D. Shukla, J. D. Wheelock, and N. Saxena, "Hidden reality: caution, your hand gesture inputs in the immersive virtual world are visible to all!" in *USENIX Security*, 2023, pp. 859–876.

[45] H. Guo, H.-N. Dai, X. Luo, Z. Zheng, G. Xu, and F. He, "An empirical study on oculus virtual reality applications: Security and privacy perspectives," in *ICSE*, 2024, pp. 1–13.

[46] G. Hernandez, D. J. Tian, A. S. Yadav, B. J. Williams, and K. R. Butler, "{BigMAC}:{Fine-Grained} policy analysis of android firmware," in *USENIX Security*, 2020.

[47] Q. Hou, W. Diao, Y. Wang, X. Liu, S. Liu, L. Ying, S. Guo, Y. Li, M. Nie, and H. Duan, "Large-scale security measurements on the android firmware ecosystem," in *ICSE*, 2022.

[48] Q. Hou, W. Diao, Y. Wang, C. Mao, L. Ying, S. Liu, X. Liu, Y. Li, S. Guo, M. Nie *et al.*, "Can we trust the phone vendors? comprehensive security measurements on the android firmware ecosystem," *TSE*, 2023.

[49] S. Ishaque, A. Rueda, B. Nguyen, N. Khan, and S. Krishnan, "Physiological signal analysis and classification of stress from virtual reality video game," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2020, pp. 867–870.

[50] I. Jarin, Y. Duan, R. Trimananda, H. Cui, S. Elmaleki, and A. Markopoulou, "Behavr: User identification based on vr sensor data," *arXiv preprint arXiv:2308.07304*, 2023.

[51] JDSUPRA, "A brave new world: How to manage safety hazards from augmented reality (ar), virtual reality (vr), and artificial intelligence (ai)," https://www.jdsupra.com/legalnews/a-brave-new-world-how-to-manage-safety-3563947/.

[52] Y. Ji, M. Elsabagh, R. Johnson, and A. Stavrou, "{DEFInit}: An analysis of exposed android init routines," in *USENIX Security*, 2021.

[53] W. Jin, Y. Dai, J. Zheng, Y. Qu, M. Fan, Z. Huang, D. Huang, and T. Liu, "Dependency facade: The coupling and conflicts between android framework and its customization," in *ICSE*. IEEE, 2023.

[54] Kaspersky, "What are the security and privacy risks of vr and ar," https://usa.kaspersky.com/resource-center/threats/security-and-privacy-risks-of-ar-and-vr.

[55] A. Kumar, L.-H. Lee, J. Chauhan, X. Su, M. A. Hoque, S. Pirttikangas, S. Tarkoma, and P. Hui, "Passwalk: Spatial authentication leveraging lateral shift and gaze on mobile headsets," in *ACM Multimedia*, 2022.

[56] Y.-T. Lee, W. Enck, H. Chen, H. Vijayakumar, N. Li, Z. Qian, D. Wang, G. Petracca, and T. Jaeger, "{PolyScope}:{Multi-Policy} access control analysis to compute authorized attack operations in android systems," in *USENIX Security*, 2021.

[57] J. Li, A. Roy Chowdhury, K. Fawaz, and Y. Kim, "Kaleido: Real-time privacy control for eye-tracking systems," in *USENIX Security*, 2021.

[58] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "Libd: Scalable and precise third-party library detection in android markets," in *ICSE*. IEEE, 2017.

[59] J. Liebers, M. Abdelaziz, L. Mecke, A. Saad, J. Auda, U. Gruenefeld, F. Alt, and S. Schneegass, "Understanding user identification in virtual reality through behavioral biometrics and the effect of body normalization," in *CHI '21*, 2021.

[60] P. Liu, M. Fazzini, J. Grundy, and L. Li, "Do customized android frameworks keep pace with android?" in *MSR*, 2022.

[61] P. I. P. Ltd., "Pico os," https://www.picoxr.com/global/software/pico-os.

[62] S. Luo, X. Hu, and Z. Yan, "Holologger: Keystroke inference on mixed reality head mounted displays," in *IEEE VR*. IEEE, 2022, pp. 445–454.

[63] 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.

[64] L. Maar, F. Draschbacher, L. Lamster, and S. Mangard, "{Defects-in-Depth}: Analyzing the integration of effective defenses against {One-Day} exploits in android kernels," in *USENIX Security*, 2024, pp. 4517–4534.

[65] M. H. Meng, Q. Zhang, G. Xia, Y. Zheng, Y. Zhang, G. Bai, Z. Liu, S. G. Teo, and J. S. Dong, "Post-gdpr threat hunting on android phones: Dissecting os-level safeguards of user-unresettable identifiers." in *NDSS*, 2023.

[66] Meta, "Meta community forums," https://communityforums.atmeta.com/.

[67] ——, "Meta quest 2: Defense through offense," https://engineering.fb.com/2023/09/12/security/meta-quest-2-defense-through-offense/.

[68] ——, "Set up your boundary for meta ques," https://www.meta.com/help/quest/articles/in-vr-experiences/oculus-features/oculus-guardian/.

[69] Meta, "Unlocking oculus go," https://developers.meta.com/horizon/blog/unlocking-oculus-go/, 2021.

[70] Meta Platforms, "Meta quest 3: Technical specifications," 2023. [Online]. Available: https://www.meta.com/quest/quest-3/specs/

[71] 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*. IEEE, 2022, pp. 765–774.

[72] R. Miller, N. K. Banerjee, and S. Banerjee, "Combining real-world constraints on user behavior with deep neural networks for virtual reality (vr) biometrics," in *IEEE VR*. IEEE, 2022.

[73] V. Nair, G. M. Garrido, and D. Song, "Exploring the unprecedented privacy risks of the metaverse," *arXiv preprint arXiv:2207.13176*, 2022.

[74] V. Nair, W. Guo, J. F. O'Brien, L. Rosenberg, and D. Song, "Deep Motion Masking for Secure, Usable, and Scalable Real-Time Anonymization of Virtual Reality Motion Data," Nov. 2023, arXiv:2311.05090 [cs].

[75] V. C. Nair, G. Munilla-Garrido, and D. Song, "Going Incognito in the Metaverse: Achieving Theoretically Optimal Privacy-Usability Tradeoffs in VR," in *UIST*. San Francisco CA USA: ACM, Oct. 2023, pp. 1–16.

[76] A. Nguyen, X. Zhang, and Z. Yan, "Penetration Vision through Virtual Reality Headsets: Identifying 360-degree Videos from Head Movements," in *USENIX Security*, 2024.

[77] NIST-CVE-Database, "Cve-2015-3864," https://nvd.nist.gov/vuln/detail/CVE-2015-3864.

[78] I. Olade, C. Fleming, and H.-N. Liang, "Biomove: Biometric user identification from human kinesiological movements for virtual reality systems," *Sensors*, 2020.

[79] K. Pfeuffer, M. J. Geiger, S. Prange, L. Mecke, D. Buschek, and F. Alt, "Behavioural biometrics in vr: Identifying people from body motion and relations in virtual reality," in *CHI '19*, 2019.

[80] A. Possemato, S. Aonzo, D. Balzarotti, and Y. Fratantonio, "Trust, but verify: A longitudinal analysis of android oem compliance and customization," in *S&P (Oakland)*. IEEE, 2021.

[81] C. Reis, A. Moshchuk, and N. Oskov, "Site isolation: Process separation for web sites within the browser," in *USENIX Security*, 2019, pp. 1661–1678.

[82] RoadToVR, "Meta has sold nearly 20 million quest headsets, but retention struggles remain," https://www.roadtovr.com/quest-sales-20-million-retention-struggles/.

[83] J. Šalkevicius, R. Damaševičius, R. Maskeliunas, and I. Laukienė, "Anxiety level recognition for virtual reality therapy system using physiological signals," *Electronics*, 2019.

[84] H. Sidhpurwala, "Hardening elf binaries using relocation read-only (relro)," https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read-only-relro.

[85] V. S. Simhadri, Y. Xiong, H. Farrukh, and X. Zhang, "Virtual reality, real problems: A longitudinal security analysis of vr firmware," 2025. [Online]. Available: https://arxiv.org/abs/2509.00662

[86] M. Slater and M. V. Sanchez-Vives, "Enhancing our lives with immersive virtual reality. frontiers in robotics and ai, 3, 74," 2016.

[87] 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.

[88] S. Smalley and R. Craig, "Security enhanced (se) android: bringing flexible mac to android." in *NDSS*, vol. 310, 2013, pp. 20–38.

[89] ssut, "payload-dumper-go," https://github.com/ssut/payload-dumper-go.

[90] Statista, "Virtual reality (vr) - statistics & facts," https://www.statista.com/topics/2532/virtual-reality-vr/.

[91] Z. Su, K. Cai, R. Beeler, L. Dresel, A. Garcia, I. Grishchenko, Y. Tian, C. Kruegel, and G. Vigna, "Remote Keylogging Attacks in Multi-user VR Applications," May 2024, arXiv:2405.14036 [cs]. [Online]. Available: http://arxiv.org/abs/2405.14036

[92] T. Sutter and B. Tellenbach, "Firmwaredroid: Towards automated static analysis of pre-installed android apps," in *(MOBILESoft)*. IEEE, 2023.

[93] D. J. Tian, G. Hernandez, J. I. Choi, V. Frost, C. Raules, P. Traynor, H. Vijayakumar, L. Harrison, A. Rahmati, M. Grace *et al.*, "{ATtention} spanned: Comprehensive vulnerability analysis of {AT} commands within the android ecosystem," in *USENIX Security*, 2018.

[94] P. P. Tricomi, F. Nenna, L. Pajola, M. Conti, and L. Gamberini, "You can't hide behind your headset: User profiling in augmented and virtual reality," *arXiv preprint arXiv:2209.10849*, 2022.

[95] 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, pp. 3789–3806.

[96] R. user. (2023) Comment on the post "can't enabled hand tracking help?". [Online]. Available: https://www.reddit.com/r/MetaQuestVR/comments/1ihblz2/comment/mavwlvw/

[97] T. Verge, "Surveillance will follow us into the 'metaverse', and our bodies could be its new data source," https://www.washingtonpost.com/technology/2022/01/13/privacy-vr-metaverse/.

[98] ——, "This is meta's ar / vr hardware roadmap for the next four years," https://www.theverge.com/2023/2/28/23619730/meta-vr-oculus-ar-glasses-smartwatch-plans.

[99] X. Wang, Y. Zhang, X. Wang, Y. Jia, and L. Xing, "Union under duress: Understanding hazards of duplicate resource mismediation in android software supply chain," in *USENIX Security*, 2023.

[100] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on android security," in *CCS*, 2013.

[101] Z. Xie, M. Wen, H. Jia, X. Guo, X. Huang, D. Zou, and H. Jin, "Precise and efficient patch presence test for android applications against code obfuscation," in *ISSTA*, 2023.

[102] XRToday, "Virtual reality security and privacy," https://www.xrtoday.com/virtual-reality/virtual-reality-security-and-privacy/.

[103] Z. Yang, Z. Sarwar, I. Hwang, R. Bhaskar, B. Y. Zhao, and H. Zheng, "Can Virtual Reality Protect Users from Keystroke Inference Attacks?" *arXiv preprint arXiv:2310.16191*, 2023.

[104] D. Yu, G. Yang, G. Meng, X. Gong, X. Zhang, X. Xiang, X. Wang, Y. Jiang, K. Chen, W. Zou *et al.*, "Sepal: Towards a large-scale analysis of seandroid policy customization," in *WWW*, 2021.

[105] R. Yu, F. Del Nin, Y. Zhang, S. Huang, P. Kaliyar, S. Zakto, M. Conti, G. Portokalidis, and J. Xu, "Building embedded systems like it's 1996," in *NDSS*, 2022.

[106] X. Zhan, L. Fan, T. Liu, S. Chen, L. Li, H. Wang, Y. Xu, X. Luo, and J. Liu, "Automated third-party library detection for android applications: Are we there yet?" in *ASE*, 2020.

[107] Y. Zhan, Y. Meng, L. Zhou, Y. Xiong, X. Zhang, L. Ma, G. Chen, Q. Pei, and H. Zhu, "Vpvet: Vetting privacy policies of virtual reality apps," in *CCS*, 2024, pp. 1746–1760.

[108] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song, and W. Zou, "Practical control flow integrity and randomization for binary executables," in *S&P (Oakland)*. IEEE, 2013, pp. 559–573.

[109] H. Zhang and Z. Qian, "Precise and accurate patch presence test for binaries," in *USENIX Security*, 2018.

[110] J. Zhang, A. R. Beresford, and S. A. Kollmann, "Libid: reliable identification of obfuscated third-party android libraries," in *ISSTA*, 2019.

[111] 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.

[112] Z. Zhang, H. Zhang, Z. Qian, and B. Lau, "An investigation of the android kernel patch ecosystem," in *USENIX Security*, 2021, pp. 3649–3666.

[113] X. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The peril of fragmentation: Security hazards in android device driver customizations," in *S&P (Oakland)*. IEEE, 2014.