Securing Virtual Reality Apps Inter-process Communication

Oluwatosin Falebita(✉), Mahmoud Abdelgawad, Evan Anspach, and Indrakshi Ray

Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA
{oluwatosin.falebita,m.abdelgawad,pidge.anspach, indrakshi.ray}@colostate.edu

Abstract. As Virtual Reality (VR) technology continues to gain widespread adoption, concerns about cybersecurity threats in immersive environments are mounting. Inter-process communication (IPC) is prevalent in operating systems that power VR devices, allowing applications to interact. While facilitating interaction between apps, the IPC presents a significant risk, as malicious apps can exploit it to attack other apps. This vulnerability is rooted in the inherent security architecture and design implementations of these operating systems. This paper addresses IPC security by proposing a fine-grained security model that employs the NIST Next Generation Access Control (NGAC). It focuses on the Android environment to safeguard the IPC and secure the apps' interaction. This security model strengthens VR systems' defensive capabilities and ensures the safety of users' digital experiences. The security model is designed as an NGAC gateway that is adaptable at the kernel level and portable to various operating system architectures. This NGAC gateway fortifies Android-based VR devices and assures the consumer VR market. Such a security model is required for the VR security domain and will be transformative for the VR industry.

Keywords: Virtual Reality · Inter Process Communication · Next Generation Access Control (NGAC) · Virtual Reality Apps · Intent Attacks · Secure App Inter-Communication

1 Introduction

Virtual Reality (VR) has started to find its foothold in modern computing. However, they have yet to be fully secured against many cyberattacks targeting VR applications (apps) and devices. VR devices and apps are prone to various cyberattacks by compromising confidentiality, integrity, and availability properties in the form of eavesdropping [36], perception manipulation [35], and denial of service (DoS) [32] attacks. Moreover, the data captured by headset sensors, such as eye tracking, are sensitive. Such data must be protected during storage and transit. Security and privacy breaches have far-reaching and devastating consequences.

V. T. Patil et al. (Eds.): ICISS 2024, LNCS 15416, pp. 63–84, 2025.
https://doi.org/10.1007/978-3-031-80020-7_4

Researchers are now uncovering security concerns in VR environments such as Side Channel Spying/Eavesdropping [6,29,31], Perception Manipulation [11,12,35,39], and Man in the Middle attacks [36]. Many of these vulnerabilities are based on the insecure architecture of the device operating system (OS). They allow an adversary to gain access to protected data (*side channel spying/eavesdropping, man in the middle, headset tracking*) or to influence the perception of the headset user while using the VR device (*perception manipulation*).

VR apps require interaction with one another and request access to various system services. Many cyberattacks exploit weaknesses in the inter-process communication (IPC) prevalent in the operating system of these headsets, particularly devices powered by a custom Android operating system [11,36,40]. Previously found attacks, such as perception manipulation, rely on insecure IPC to manipulate virtual content by modifying configuration files and sensor settings using system calls.

The traditional access control mechanisms, such as discretionary access control (DAC) and mandatory access control (MAC), are enforced through the Security Enhanced Android (SEAndroid) [34], a customized version of SELinux used by the Binder IPC [34]. The Binder IPC mechanism allows components within a single process or across separate processes to communicate with one another in the Meta Horizon operating system [26], a custom version of VR operating system built on Android. Meta Quest VR devices are powered by Meta Horizon OS and relies primarily on Intent firewall and Intent filters to secure inter-process communication between apps. These mechanisms failed to identify malicious intent specifically when an implicit intent (a messaging object that allows an app to declare a general action to perform, which allows a component from another app to handle it) is utilized [2], resulting in intent spoofing or intent manipulation attacks on the device.

Meta Horizon OS utilizes DAC and MAC. DAC provides a user-centric model [23] where the user has the discretion to allow apps access to sensitive resources like the camera, microphone, or contacts through permission requests. However, this can lead to security risks if users are not careful about what permissions they grant. The OS implements MAC based on classifications and security labels through SEAndroid to enforce strict policies at the system level. However, these access control policies are determined by the lax firewall rules in the Intent firewall, which introduces vulnerabilities in the IPC. The current LSM hooks [34] Binder IPC is protected by context labels policies, which are matched with Intent Filters and the permission specified by the app developer. If none exist, such exported component remains vulnerable [2,4].

We therefore focus on NIST Next Generation Access Control (NGAC) to handle traditional access control's limitations. The NGAC has been standardized by the National Institute of Standards and Technology in alignment with the American National Standards Institute/International Committee for Information Technology Standards (ANSI/INCITS); refer to the standard "INCITS 565-2020" [21]. NGAC is a generic architecture that defines access control as a

reusable set of data abstractions and attribute-based functions. It is suitable for expressing access control policies for a wide range of applications, including those spanning multiple distributed, interconnected processes and situational monitoring applications in which policies may be changed while they are deployed.

While several works [18,29,31] focus on applying new authentication schemes, encryption and other modular security on top of the existing operating system architecture [19], our work focuses on augmenting the existing framework by introducing NGAC, which is based on the attributes of the interacting apps. Adding NGAC as another authorization layer for IPC verification increases the efficiency of securing apps' interaction. This solution allows granular communication verification between apps to prevent malicious apps from performing malicious intent manipulation attacks on legitimate apps.

Key Contribution: This work has three contributions described as follows:

1. We identify the major access control attacks in a virtual reality environment.
2. We formulate access control policies, adhering to the NGAC model, to secure inter-process communication and illustrate how they can be enforced in virtual reality devices.
3. We develop a policy enforcement architecture suitable for Android-based virtual reality devices.

The rest of the paper is organized as follows. Section 2 discusses the background needed to understand this work. It covers details of the VR apps IPC and their security flaws. It also provides some details about the NIST NGAC. Section 3 describes the policy design for validating IPC in an Android environment. Section 4 provides a secure IPC architecture model using NGAC. Section 5 discusses attack mitigation through the NGAC security model. Section 6 explores the related work on VR security and Android IPC security. Finally, Sect. 7 summarizes our work and points to future plans.

2 Background

This section provides an overview of the VR app IPC structure, VR app IPC attacks, and NGAC model.

2.1 VR Apps Inter-process Communication

Meta Horizon OS is a modified version of Android [33]; it incorporates key Android features, notably the Binder, which is the primary IPC channel that allows apps and system resources to interact. Most Android-based VR operating systems utilize sandboxing to prohibit direct access to the resources of other apps and the Android system [20]. However, an app communicates with other apps and uses system services through the Binder using the following mechanisms.

Activity provides a user with an interface to interact with an app. An app can permit another app to launch its activities.

Broadcast services enable the system to transmit events to apps outside the typical user flow, allowing the apps to respond to system-wide broadcast announcements. A broadcast receiver operates in the background to manage incoming events.

Content providers are responsible for storing the app's data and facilitating the app's data sharing with other apps' components.

Services provides an entry point for keeping and running an app in the background without a user interface while performing various computational tasks.

VR app's inter-process communication utilizes the Binder driver and SELinux context labels to control access and determine which processes can communicate with other app components using the following essential components of the Binder:

Binder Driver: A kernel-level component that acts as the core of the Binder IPC mechanism. It manages the communication between processes and enforces security policies based on predefined SElinux context(label) consisting of user, role, type, and level [34]. The Binder driver oversees the creation of Binder objects, the passing of references, and the delivery of transactions. The Binder driver ensures that messages are securely and efficiently routed between processes.

Service Manager: It is the central registry for all Binder services in the system. When a service is created, it registers itself with the Service Manager, allowing other processes to discover and interact. The Service Manager acts as a lookup table for Binder services, enabling clients to find the Binder objects when they need to communicate.

Native Object: It defines an interface for the services a target process or an app provides. It serves as a concrete class that does the actual work of receiving calls from a client, processing those calls, and returning the results [8]; it creates a unique identifier called Binder token, which helps the Binder driver track and manage Binder references across different processes.

Proxy Object: A proxy object marshals method calls from the client process, sending them through the Binder framework to the actual service object in the service process.

A *Caller app* can interact with a *Target app* by specifying an *Intent*. An intent is a messaging object used to request an action from another app component. As shown in Fig. 1, the flow involves several steps and components.

Step 1 - Service Discovery: The caller app queries the service manager to check if any app offers the requested service. This requires the caller app to specify an intent for the operation it needs to be performed [1]. The service manager uses this intent to look up an appropriate service; if the result is null, then there are no apps on the device that can receive such an intent, and an app crash occurs with an error message, and the request will be terminated; if the result is not null, then at least one app can handle the request.

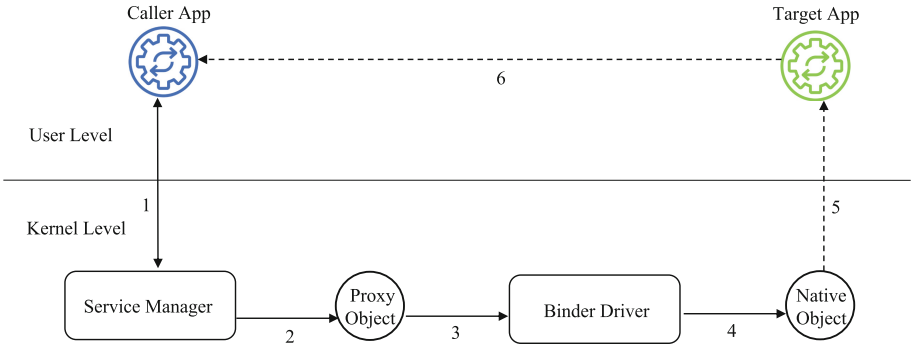

Fig. 1. VR Apps Binder Inter Process Communication

Step 2 - Proxy Object Creation: Upon receiving a not null result, the caller process creates a proxy object (node) that contains the intent to be performed on the target app.
Step 3 to 5 - Parcel copy: In steps 3 to 6, the binder driver copies data from the proxy object to the target app via the native object, preserving its structure and layout while performing the operation.
Step 6 - Receiving the Response: The target app returns the result of the request to the caller app.

2.2 VR Apps IPC Attacks

Once a VR app is installed, the operating system catalogs the app's declared services in the Service Manager to keep track of the intents that the app can handle. When a caller app initiates an activity using startActivity() or startActivityForResult() with an implicit intent, the system searches for compatible activities [3] that can perform such intent without checking for proper access control [15]. These inadvertently create potential pathways for unauthorized access such as immersive attacks [11] and immersive hijacking [39].

Immersive attacks [11] are defined as attacks that target the immersive nature of VR devices in order to exploit vulnerabilities in the perception of the immersed user. A successful immersive manipulation attack will result in a virtual environment modified to disrupt a user's task or cause physical or mental harm to the user. This work outlines four immersive attacks: the Chaperone Attack, the Disorientation Attack, the Human Joystick (Redirected walking) Attack, and the Overlay Attack. These attacks were performed on systems running OpenVR, for example, HTC Vive and Oculus Rift VR, by taking advantage of a weakness in the IPC framework's inability to determine whether the caller application is benign or malicious and whether it should be allowed to perform the requested operation(s) on the target application or system resources.

Immersive hijacking [39] presents a form of access control violation specifically targeting Meta VR platforms. This attack exploits weak access control in

the operating system's IPC, which is crucial for facilitating various app interactions. These interactions range from launching other apps to sending and receiving data through intents. In a more specific instance, immersive hijacking targets the Meta Horizon Operating System (OS), a VR OS developed by Meta. This attack method cleverly exploits improper validation and access control in the IPC mechanism to create a deceptive layer of virtual reality within Meta OS, effectively generating a nested reality. The attack begins with side-loading a seemingly benign VR Home screen application onto a device and a spy script that runs in the background. When the user signals the system to exit the current application, expecting to return to the home environment, the spy script intercepts and destroys the signal, terminates the application, and initiates the malicious VR Home screen. This allows attackers to perform arbitrary operations, such as intercepting and modifying requests sent to other applications.

A significant area for security improvement in this setup is the device's operating system's insufficient IPC access control and validation process, which overly depends on permissions as determined by app developers. Often, these permissions are not strictly enforced, placing an undue security burden on the developers, with only minimal support from the operating system security framework.

2.3 NIST Next Generation Access Control (NGAC)

NGAC is a revolutionary approach to managing and implementing access control mechanisms [16]. It emphasizes more detailed, flexible, and context-aware policies compared to traditional models such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-Based Access Control (RBAC). Each of the components of NGAC is specifically formulated to adapt to the changing security requirements of contemporary digital settings, encompassing Blockchain [27], Internet of Things (IoT) [10], and mobile devices.

NGAC utilizes the policy class, which can be conceptualized as a hierarchical graph consisting of a predetermined set of relations and access rights between policy components. It is used for making access control decisions using reference mediation and contains elements such as authorized users, access rights, objects, processes, and operations. The policy class defines several relationships among elements, including assignment, association, prohibition, and obligation relations [17].

- **Assignment** relation is used in expressing relationships between policy elements such as the relationship between users and user attributes, user attributes to other user attributes, resources to resource attributes, resource attributes to other resource attributes, user attributes to a policy class, and resource attributes to a policy class.
- **Association** relation establishes the authorization of access rights between policy elements such as user attributes and resource attributes. It determines which users can access specific resources and exercise specific access rights.
- **Prohibition** relations entail denying access rights between policy elements, such as user characteristics and resource attributes. It consists of two primary

prohibitions: user deny and process deny, which prohibits access to user and process respectively.
- **Obligation** can be employed to automatically adjust policies based on certain conditions related to access modes and patterns triggered by the execution of events. An obligation can be defined by two essential elements: an event pattern and a corresponding response. Our work considers controlling the IPC between apps. Thus, the obligation relations are omitted from the NGAC graph.

NGAC consists of five process points: Policy Decision Point, Policy Enforcement Point, Policy Administration Point, Event Processing Point, and Resource Access Point [9]. These policy points are briefly described as follows.

- **Policy Decision Point (PDP)** This is the core component responsible for making access control decisions based on predefined policies. The PDP evaluates access requests against the policies and determines whether to grant or deny the requests based on the applicable rules.
- **Policy Enforcement Point (PEP)** This component is responsible for enforcing the decisions made by the PDP. It acts at the point of access, ensuring that only authorized actions are allowed to proceed based on the decision provided by the PDP.
- **Policy Information Point (PIP)** The PIP stores attributes related to users, resources, and environmental conditions. The PDP uses these attributes to make informed access control decisions. The PIP can pull information from various sources, including directories, databases, and other external sources.
- **Policy Administration Point (PAP)** This component creates, manages, and deletes access control policies. The PAP provides an interface for administrators to define and modify the rules and policies that the PDP evaluates.
- **Event Processing Point (EPP)** serves as a critical functional component within a system. It operates by receiving event contexts and comparing them against predefined event patterns that are part of obligations. When a match is found, the EPP forwards the corresponding event responses to the Policy Decision Point (PDP).
- **Resource Access Point (RAP)** is a crucial functional entity designed to be the sole gateway for accessing designated protected resources. This exclusive access point ensures all interactions with these resources are centrally controlled and managed.

3 Policy Design for Securing VR Apps IPC Using NGAC

Virtual Reality Operating Systems (VROS) are complex environments that require robust access control mechanisms to ensure secure and authorized interactions within the virtual reality space. By leveraging the NGAC framework, we can design flexible and extensible policies that govern access to various components and functionalities of the VROS.

Using the NGAC approach, we aim to create a robust and scalable access control framework for VROS. This framework will enable precise control over resource access and virtual interactions while promoting security, compliance, and operational efficiency within the virtual reality space. The NGAC standard provides a logical, attribute-based approach to defining relationships between users (subjects), resources (objects), and the operations subjects can perform on objects. In the context of a VROS, subjects may represent virtual reality users, applications, or system processes, while objects can encompass virtual environments, system resources, or simulated resources.

Our current work focuses on controlling inter-processing communication. Consequently, in our model, caller apps are the subject, and target apps are the objects. The NGAC we propose will manage the access between caller apps and target apps, verifying their attributes and checking the privileges of the NGAC model to grant or deny access.

We identify the attributes of the caller apps and target apps within the VROS environment. Once the necessary attributes are defined, we build the NGAC model to fulfill the security policy requirements that dictate the conditions under which caller apps can access, manipulate, or interact with the target apps' virtual reality resources and functionalities. Throughout the security policy design process, we adhere to the NGAC standard's guidelines and best practices, ensuring that policy requirements are logically sound, unambiguous, and enforceable within the VROS environment. Additionally, we design security policies that are flexible and extensible, allowing for future modifications or additions as the VROS system's requirements evolve.

3.1 Identifying Policy Element of IPC

Policy elements are the fundamental components of the NGAC policy class, which include authorized subjects (S), processes (P), objects (O) operations (Op), and access rights (AR) [17]. The policy class comprises a finite set of relations between these elements used to grant or deny an access request. We determine operating system entities relevant to virtual reality and compile a list of the critical attributes required to define our policy class.

Subject: This is the caller app that requests a service from the target app. A caller app can be a user-installed application (third party) or a pre-installed application (native application). A caller app can request access to the components of another app through implicit intent and may require access to various system resources within the operating environment. We define a list of attributes CA as a set of finite attributes belonging to the caller app, described as:

For simplicity, we use the term "signed" to describe apps that have been signed by trusted entities such as Google Play, Oculus Store, or a device manufacturer's store. "unsigned" refers to apps not signed by these entities or those signed by untrusted sources. A "trusted source" refers to applications downloaded from a trusted entity. At the same time, "sideload" refers to apps installed on a device from other sources, such as the internet or external methods (e.g., ADB or file sharing).

1. *CA_sig* refers to caller app signature which is of the enumerated type [signed, unsigned].
2. *CA_is* is the caller app installation source, which is of the enumerated type [trusted source, sideload].

Object: The target app refers to the an application or system resource from which the caller app requests a service; the target app provides a finite set of services listed in the manifest file to the service manager. We define a list of attributes TA as a set of finite attributes belonging to the target app, described as:

1. *TA_sig* = target app signature which is of the enumerated type [signed, unsigned]
2. *TA_is* = target app installation source [trusted source, sideload]
3. *TA_cat* = target app type [non-native app, native app, system resources]

Access Rights: The caller app can specify an operation to perform on the target app based on the components provided. These operations are considered access rights in the context of access control. Examples of possible operations are listed below.

- Activity = startactivity, finish
- Services = bindservice, unbindservice
- Broadcast receivers = sendbroadcast, registerreceiver

3.2 NGAC IPC Policy Design

Virtual reality operating system includes various native apps (browser and photos) and system resources (camera, location, microphone, and haptic controller). It also includes non-native apps such as business, gaming, and entertainment. These apps need to be protected from malicious apps. The trusted apps are usually installed and signed by a trusted source (e.g., Meta Horizon store) or can be self-signed by the developer for development purpose and in such a case we consider the signature as unsigned; these trusted apps are allowed to perform operations such as *startActivity*, *bindService*, and *unbindService* on system resources and other apps. However, apps installed from untrusted sources (i.e., sideload) are suspicious. If the sideload apps are signed, we want to allow only specific operations to be executed on the system resources. Meanwhile, if the sideload apps are unsigned, there is a higher probability that they are malicious apps; in these cases, we want to stop or deny any operations on the system resources and other apps. Table 1 represents these restrictions of inter-process communication between VR apps in three security policies.

PC1 PC2, and PC4 show association relations between caller and target apps. However, PC2 grants only one operation, while PC1 and PC4 grants all operations. PC3 represents a prohibition relation, denying all operations on system resources and other target apps. These security policies are analyzed and

Table 1. IPC Policy

Policy ID	Policy Description
PC1	if caller app is from a trusted source and signed, it is granted permission to perform any operations on any signed target app.
PC2	if caller app is from a trusted source and unsigned then it can perform only *startActivity* operation on target app (system resources)
PC3	if caller app is sideload and unsigned, then deny all operations on any target app
PC4	if caller app is from a trusted source and signed, it is granted permission to perform any operations on unsigned target app.

transformed into the NGAC security model, an authorization graph shown in Fig. 2. Each policy represents one or many paths from caller apps through caller app attributes, associations or prohibition relations, and target app attributes to the target apps. These attributes have various values. The attributes, possible values, relations, and operations are described below.

- Activity = startActivity, finish
- Services = bindService, unbindService
- Broadcast receivers = sendBroadcast, registerReceiver

- **Basic elements:**
  - Caller app = {Custom App, Oculus Browser, Horizon Edge}
  - Caller app attributes = {*CA_sig, CA_is*}
  - Target app = {GoMeet, Photos }
  - Target app attributes = {*TA_sig, TA_cat*}
  - Access right = {startActivity, finish, bindService, unbindService, sendBroadcast, registerReceiver, query, insert, All}
  - Policy Class = {Intent Validation}

- **Relations:**
  - Assignment ={(Custom App, Side load), (Oculus Browser, Trusted Source), (Horizon Edge, Trusted Source), (GoMeet, Non-native app), (Photos, Native app) (Sideload, Unsigned), (Sideload, Signed), (Non-native app, Signed), (Native app, Signed) (System resources, Signed), (Unsigned, Intent Validation), (Signed, Intent Validation)}
  - Association = {(trusted, Grant[bindService], System resource), (Signed, Grant[All], Signed)}
  - Prohibitions = {(Unsigned, Deny[All], Signed)}

We use Depth-first search (DFS [25]) to derive the privilege list from the NGAC model. The privilege list states each caller app and its privileges (grant/deny) to perform operations on a target app. The privilege list is formatted as privilege list = [(caller App, [caller attributes], permission[access rights], [target attributes], target App)].

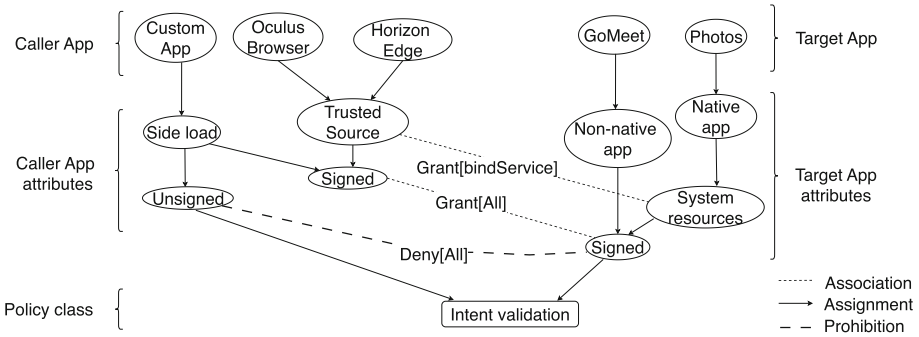

Fig. 2. NGAC Authorization Graph

Next, we formally define the *intent* as a set of NGAC basic elements (attributes) and design an algorithm that validates it against the privilege list to grant/deny inter-process communication between apps.

4 NGAC IPC Policy Architecture

The policy class for inter-process communication (IPC) with implicit intent is to guarantee that interactions between the caller app and the target app are validated before being executed on the target app. The validation process first intercepts an intent and then verifies the caller app signature and installation source. It also verifies the target app signature and type.

Definition 1 (Intent). *is defined as 6-tuple* $INTENT = \langle \mathcal{C}, \mathcal{T}, \mathcal{CA}, \mathcal{TA}, \mathcal{OP}, \mathcal{E} \rangle$*, where*

- $\mathcal{C}$ *is a finite set of caller apps;* $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$.
- $\mathcal{T}$ *is a finite set of target apps;* $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$.
- $\mathcal{CA}$ *is a finite set of caller app's attributes;* $\mathcal{CA} = \{ca_1, ca_2, \ldots, ca_n\}$.
- $\mathcal{TA}$ *is a finite set of target app's attributes;* $\mathcal{TA} = \{ta_1, ta_2, \ldots, ta_n\}$.
- $\mathcal{OP}$ *is a finite set of access rights (operations) declared by the NGAC administrator;* $\mathcal{OP} = \{op_1, op_2, \ldots, op_n\}$
- $\mathcal{E}$ *is a function forms intents;* $\mathcal{E} = \{(c, [ca], [op], [ta], t) \in c \times \mathcal{CA} \times \mathcal{OP} \times \mathcal{TA} \times t \mid c \in \mathcal{C}, ca \in \mathcal{CA}, op \in \mathcal{OP}, ta \in \mathcal{TA},$ *and* $t \in \mathcal{T}\}$

For simplicity, we derive two lists: grant privilege and deny privilege list.

1. The grant privilege list generated from our model is as follows: (Horizon Edge, [trusted], Grant [bindService], [System resource, Native Apps], Photos), (Oculus Browser, [trusted, Signed], Grant[All], [Signed, Non-native Apps, System resource, Native Apps], GoMeet), (Oculus Browser, [trusted, Signed],

Algorithm 1: Intent Validation

Input : $Intent = \langle \mathcal{C}, \mathcal{T}, \mathcal{CA}, \mathcal{TA}, \mathcal{OP}, \mathcal{E} \rangle$
Input : $GrantPrivileges = [(callApp, [CallarAttrs], Grant[AR], [TargetAttrs], targetApp)]$
Input : $DenyPrivileges = [(callApp, [CallarAttrs], Deny[AR], [TargetAttrs], targetApp)]$
Output: $\{Grant, Deny\}$

/* validate intent against deny privilege list */
1 **foreach** $privilge \in DenyPrivileges$ **do**
2 | **if** $(privilge.callerApp = c \in Intent.C) \wedge (privilge.targetApp = t \in Intent.T) \wedge (match(privilge.CallarAttrs, CA) \wedge (match(privilge.TargetAttrs, TA) \wedge (match(privilge.AR, OP)$ **then**
3 | | **return** $Deny$
4 | **end**
5 **end**
/* validate intent against grant privilege list */
6 **foreach** $privilge \in GrantPrivileges$ **do**
7 | **if** $(privilge.callerApp = c \in Intent.C) \wedge (privilge.targetApp = t \in Intent.T) \wedge (match(privilge.CallarAttrs, CA) \wedge (match(privilge.TargetAttrs, TA) \wedge (match(privilge.AR, OP)$ **then**
8 | | **return** $Grant$
9 | **end**
10 **end**
11 **return** $Invalid$

Grant[All], [Signed, Non-native Apps, System resource, Native Apps], Photos), (Custom App, [Side load, Signed], Grant[All], [Signed, Non-native Apps, System resource, Native Apps], GoMeet), (Custom App, [Side load, Signed], Grant[All], [Signed, Non-native Apps, System resource, Native Apps], Photos)

2. The deny privilege list is: (Custom App, [Side load, Unsigned], Deny[All], [Signed, Non-native Apps, System resource, Native Apps], GoMeet), (Custom App, [Side load, Unsigned], Deny[All], [Signed, Non-native Apps, System resource, Native Apps], Photos).

The intent definition is used as input for Algorithm 1 along with the grant privilege and deny privilege lists generated from the NGAC model. Algorithm 1 iterates brute-forcibly over these privilege lists to find a match with intent parameters. From lines 1 to 5, it iterates over the deny privilege list. Each privilege instance is compared with the intent. If the caller apps attributes and target apps attributes matches the deny privilege list, and the operations required match the access rights, it returns "Deny". Otherwise, it iterates over the grant privilege list, lines 6 to 10, and does the same comparison; if it finds a match, it returns "Grant". If no match is found in both lists, line 11, it returns "Invalid". Algorithm 1 gives deny more precedence over the grant privileges; if the deny and grant occur at the same time (i.e., a conflict policy), it gives precedence to deny.

4.1 NGAC IPC Enforcement

NGAC IPC security module is designed as a library to secure IPC, and access control policies modification are restricted to the VR device manufacturers. Once

an app is installed, the NGAC framework extracts the app attributes into the Policy Information Point.

Our fine-grained access control enforcement follows steps 1 to 13 to validate inter-process communication between apps, as shown in Fig. 3.

Supposing a caller app (Custom App) needs to launch the target app (GoMeet) by requesting an activity (e.g., startActivity) provided by the target app (GoMeet). We describe the access control steps as follows:

Step 1: Service Discovery
The caller app specifies an intent containing the intended action (android. intent. action.VIEW), category(android.intent.category.LAUNCHER), and the data (Uri. parse("package:com.gomeet.app")) required to perform the intended request. This intent is then forwarded to the service manager to identify the target app that can handle the caller app intent.

Step 2: Proxy Object creation
If the result is null, no apps on the device can handle such intent, and an app crash occurs with an error message, terminating the request. If the target app is found, as in our case, a node (proxy object) will be created.

Step 3: Entry to NGAC security model
The service manager conveys this proxy object, and the identified target app ID to the PEP for further verification and access control decision.

Step 4: Initiation of Access Request
PEP initiates an access request on behalf of the caller app attempting to access the target app. The PEP collects relevant information, such as the ID of the caller app, the target app ID, and the proxy object that contains the intent.

Step 5: Attributes query
The PDP queries the PAP for information about the caller and target apps to drive the access decision.

Step 6: Attributes retrieval
The PAP validates the query to fetch the caller app and target app attributes contained in the PIP.

Step 7: Policy evaluation
PAP returns the attributes to the PDP for policy evaluation.

Step 8: Policy decision process
The PDP retrieves the relevant policies from the policy class as contained in Table 1 and traverses the authorization graph, Fig. 2, to evaluate the request and decides whether to grant or deny the request made by the caller app and send its decision to the PEP.

Step 9: Decision communication and logging
If the response is granted, the PEP thoroughly resolves the proxy object to the RAP and logs the intent simultaneously to EPP; otherwise, it returns deny, ensuring a comprehensive validation.

Step 10: Exit from NGAC security model
The resource access point forwards the intent to the Binder Driver.

Step 11 to 12: Parcel Copy
The Binder Driver resolves the intent to the target app through the Target app native object to perform the operation on the target app.
Step 13: Receiving the Response
The target app returns the result of the request to the caller app.

This process occurs whenever an app attempts to access the services offered by another app or any other protected resource, ensuring a well-documented access history.

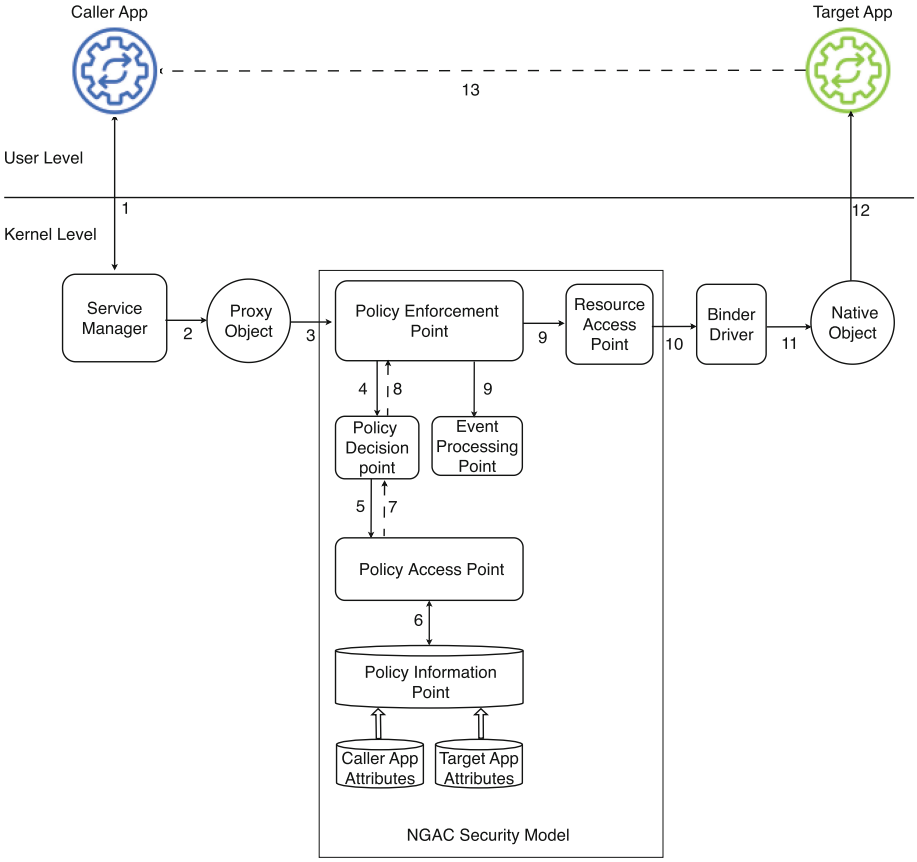

Fig. 3. NGAC IPC Enforcement Architecture

4.2 Portability to Meta Horizon Operating System

The IPC within the Meta Operating System occurs within the Android Generic Kernel Image (GKI). The IPC in the GKI utilize SEAndroid for access control

which does not have enough validating mechanism to distinguish legitimate IPC requests. To provide validation, our model replaces the standard IPC in that kernel with an IPC model that is augmented with the NGAC-IPC process described in Fig. 4. This NGAC-IPC then operates at the kernel level of the meta device in the middle of the normal Android IPC process, managing how the different processes interact and access shared services. When a new app is registered, the NGAC IPC is given its shared resources and services to make decisions regarding requests.

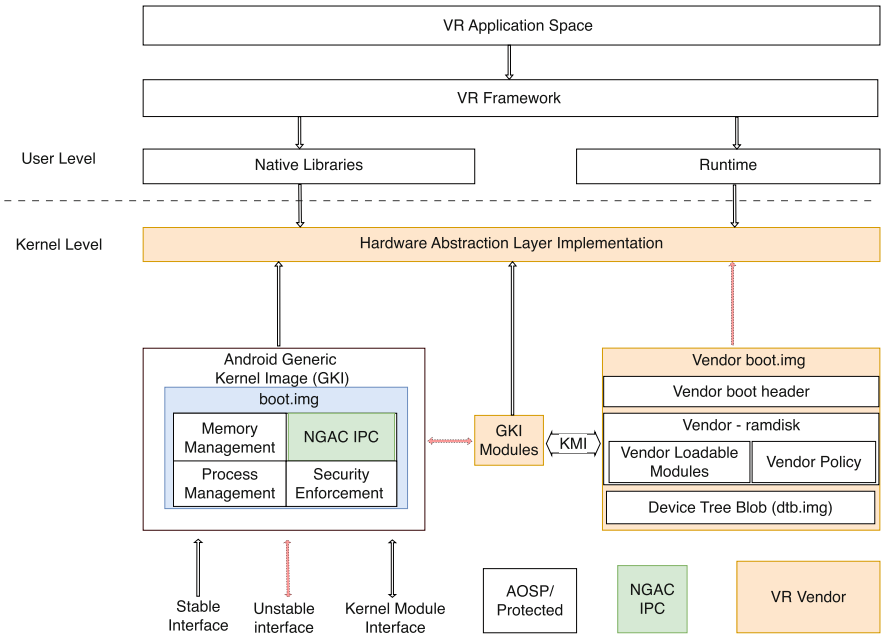

Fig. 4. NGAC IPC Portability with Meta Horizon OS

5 Discussion

We designed the policy class based on policy requirements (PC1, PC2, PC3, and PC4) to incorporate the caller app's attributes, including signature and installation source, and the target app's attributes, such as native apps, non-native apps, and system resources. These attributes are extracted from the app's manifest, and the intent is validated through policy processes to decide whether an operation should be allowed. The policy validation effectively serves as a shield against various attacks:

– Intent Hijacking: This attack occurs when an app intercepts an intent for another app and performs unauthorized operations. The policy PC3 prevents

intent hijacking attacks by ensuring intent communication is denied from caller apps that are side-loaded and unsigned or installed from a trusted source but unsigned (i.e., signed with a developer key) from accessing other apps or system resources.
- Intent Spoofing: In this attack, a malicious app generates a faked intent that looks to be from a legitimate app, leading the target app to do unwanted activities. The policy PC1 prevents such attacks by ensuring proper validation before forwarding intent to the binder.
- Intent Fuzzing: This attack sends malformed or random intents to an app to cause unexpected behavior or crashes. The policies PC1, PC2 and PC4 reduce the risk of intent fuzzing attacks by validating the operation and ensuring it has the relevant attributes to perform the operation.
- Privilege Escalation: Some intent attacks attempt to escalate privileges by exploiting flaws in how apps process intents. The policies PC1, PC2, PC4 ensure that caller apps can only communicate with specified target apps.
- Data Leakage: Malicious apps may attempt to access or extract sensitive data from other apps using intent-based attacks. The policy PC3 enforces tense validation to prevent unauthorized data access or leakage.

6 Related Work

The literature on VR security explores the vulnerabilities of VR platforms. It offers various solutions, operating system access restriction [39], Role-based Access Control (RBAC) [37]. However, these works are either limited to specific VR platforms [37] or lack the usability of the device's resources [39].

Our work addresses this research gap by safeguarding the inter-process communication in VR environments using the NIST Next-Generation Access Control (NGAC). NGAC IPC runs in the background to monitor system events. NGAC IPC's validation process, which intercepts intents and activates authorization processes, instills confidence in its effectiveness. NGAC's effectiveness in allowing the system administrator (VR device manufacturer) to add, modify, and remove policies invests a sense of security and control in safeguarding inter-process communication. This provides flexibility for securing VR apps in the Android environment. Our solution is also independent of any VR platform, providing a portable gateway that ensures VR app protection.

6.1 Related Work on VR Security

Android's Binder IPC uses a mandatory access control system named Intent Firewall (IFW) in version 4.4 [24] and relies heavily on firewall rules within the Intent Firewall [5]. It utilizes intent filters to resolve intents between interacting apps based on the permission levels set by app developers for each exported app component. When a developer fails to specify a permission level, the app component becomes exposed to intent-based attacks [24]. The operating system does not perform further validation to verify the origin of the calling app or the

legitimacy of the requested action. This leads to unauthorized activity launches and the theft of arbitrary files through activity and intent redirection attacks. The adoption of Android Binder IPC in Virtual reality devices introduced a new form of attack vector. Several access control mechanisms have been applied to virtual reality devices in the past, with differing benefits and tradeoffs using different approaches.

Wei *et al.* [37] designed a content-oriented access control mechanism for the VR platform *OpenSim* that combines identity-based capabilities and RBAC to address the security issues that are faced by systems consisting of databases, servers, and one or more clients. However, this work was limited to the *OpenSim* platform and was not extended to the device to authorize processes or users running on the headset.

Casey *et al.* [11] exposed significant IPC vulnerabilities in the HTC Vive and Oculus Rift VR devices by exploiting the software interface level. They leverage vulnerabilities within OpenVR, a software library built on StreamVR, to create and execute malicious implicit intents that the IPC mechanism failed to prevent, leading to immersive attacks by modifying configuration files. The attack resulting from these malicious implicit intents was able to guide and control a VR user's actions and movements remotely without the user's knowledge, including built-in safety mechanisms such as the chaperone, which is a boundary that prevents the immersed user from venturing into unsafe physical locations specified by the user. To mitigate these attacks, they recommended implementing application signing as a security measure to prevent IPC attacks that result in unauthorized configuration file modifications. They also suggested using the Arya framework [28] to enhance security within existing VR systems by imposing policy on sensor inputs through input stream recognizers and visual output through virtual object abstraction. However, the offered solutions must address these vulnerabilities at the operating system level.

A further study by Yang *et al.* [39] recommended suggestions such as disabling app calls by non-system apps, app certificates, validating the authenticity of app calls, or preventing user access to the OS shell to prevent IPC attacks such as immersive hijacking and interception attacks. These recommendations limit the usability of the devices.

6.2 Related Work on Android IPC Security

Several works has been done in security Android IPC, Smalley *et al.* [34] protect the Binder by implementing new Linux Security Module (LSM) security hooks in the binder driver to enforce SELinux permission checks over inter-app communication and control operations, thereby restricting which processes can communicate and manage Binder references but could not protect against malicious intent.

IPC provenance was first introduced by Dietz et al. [14]. The author developed an approach (namely Quire) to providing provenance information on Android through explicit call-chain transmission, which introduces significant developer dependency, requiring them to modify and recompile their apps

actively. This creates barriers to adoption, limits its ability to monitor existing apps, introduces potential performance issues, and exposes the system to trust and security risks based on developer compliance.

Backes *et al.* [7] proposed a system-centric solution (namely Scippa) for IPC provenance by tracking IPC calls at the system level, focusing on the Binder communication channels between apps. While this approach captures the sequence of IPC calls between apps it does validate the specific intent, data type, or action.

Kaladharan *et al.* [22] introduce the concept of a "Man-in-the-Binder" attack, which attack targets the Binder IPC mechanism as it intercepts data sent between Android applications and services, which are transmitted in plaintext, thus exposing sensitive application information. The authors propose an encryption scheme to secure the communications between them before passing the data through Binder. The encryption mechanism involves assigning a secret key between the kernel and each application or service. Although the author proposes lightweight cryptography to mitigate the performance, there may still be concerns about the impact on system performance, especially for applications that make frequent Binder calls.

Kraunelis *et al.* [26] modified the Binder library to detect and counter deceptive user interface attacks such as malicious activity launched on the Android via inspection and analysis of inter-process communication transactions in the operating system by recording application UIDs and timestamps whenever the *StartActivity* method is called. The recorded timestamp is then compared to the previously logged timestamp; if the time difference is less than a certain threshold, the UID of the previous caller is the launcher, and the current caller's UID does not own the package name. The activity is determined to be malicious, and action can be taken to mitigate the malware.

In a more recent work [30], Lyvas *et al.* explored encrypting transmitted intent data based on user-defined policies. Their approach generates a public/private key pair and a symmetric key via the Android Keystore for authentication and encryption. When an application starts an activity using an implicit intent, the system retrieves the source's keys, signs the intent with the private key, and encrypts it with the symmetric key. This allows users to control IPC by verifying the signature and decrypting the intent data. However, this system introduces overhead, with the highest measured cost being 190ms, depending on the data size.

Matteo De Giorgi [13] developed a tool called Ptracer to monitor system calls and inter-process communications in Android. Ptracer operates between the app and the kernel, intercepting and collecting data such as stack backtraces and system call parameters. It aims to detect debuggers, identify anomalies, and flag privacy concerns by analyzing how frequently an app requests sensitive data through kernel queries, focusing on dangerous behaviors related to privacy.

D. Wu *et al.* [38] introduce *SCLib*, a defense mechanism against component hijacking in Android applications. Operating at the app level, SCLib handles deployment without requiring system-level changes, making it adaptable across a fragmented Android ecosystem. It enforces MAC policies to verify

incoming requests before execution, preventing unauthorized access to exported components, mitigating privilege escalation via permission hijacking, checking system-only broadcasts, and guarding against SQL injection attacks on content providers. However, it doesn't cover all attack vectors, such as unauthorized intent receipt by malicious apps.

7 Conclusion and Future Work

This paper addresses immersive attacks in Android-based Virtual Reality devices, namely immersive manipulation and hijacking attacks, due to improper process validation during VR app interactions. We designed an enforcement architecture, called Inter-Process Communication (IPC) enforcement, that unitizes NGAC access control to manage inter-process communication between apps and protect the Android environment from malicious apps attempting to access resources and other legitimate apps.

We formally defined the apps' attributes and relations required to build the NGAC model. We also designed an algorithm to validate these attributes and relations against apps' access requests to grant/deny inter-process communication. The implementation of the IPC enforcement is a work in progress, and efforts are ongoing to bring it to production. Future work will present the implementation of IPC enforcement and experiment with its effectiveness. We are confident that this work, including implementation, will be beneficial to the VR industry.

Currently, our approach focuses on compatibility with Android-based VR devices. It does not address the security concerns on the entire range of VR devices in the current commercial market or devices outside the VR space, such as AR devices or computers. Such devices may also be vulnerable to exploitation through IPC-related attacks and would benefit from extending this model into their domain space. This will be a part of our future works.

We plan to extend this model to provide more specific and secure IPC on Android devices and provide their implementation. We also would like to investigate applications of this model onto different IPC structures in different operating infrastructures, such as HTC Vive a Windows-based VR device to understand the differences in the application of NGAC on these devices' IPCs. This approach may also apply to other systems outside of VR with vulnerabilities in their IPCs; this work may be extended to Kubernetes in its node communications or to other Linux systems to provide a security layer surrounding process intents.

Acknowledgements. This work was supported in part by funding from NIST under Grant Number 60NANB23D152 NSF under Award Numbers CNS 1715458, DMS 2123761, CNS 1822118, NIST, ARL, Statnett, AMI, NewPush, and Cyber Risk Research.

References

1. Android, D.: Common intents (2024). https://developer.android.com/guide/components/intents-common
2. Android, D.: Intents and intent filters (2024). https://developer.android.com/guide/components/intents-filters
3. Android, D.: Let other apps start your activity (2024). https://developer.android.com/training/basics/intents/filters
4. Android, D.: Permissions (2024). https://developer.android.com/guide/topics/manifest/permission-element
5. Android Source, C.: Interact with other apps (2023). https://cs.android.com/android/platform/superproject/+/android14-qpr3-release:frameworks/base/services/core/java/com/android/server/firewall/IntentFirewall.java;l=58?q=intentFire&ss=android%2Fplatform%2Fsuperproject
6. Arafat, A.A., Guo, Z., Awad, A.: VR-spy: a side-channel attack on virtual key-logging in VR headsets. In: 2021 IEEE Virtual Reality and 3D User Interfaces (VR), pp. 564–572 (2021). https://doi.org/10.1109/VR50410.2021.00081
7. Backes, M., Bugiel, S., Gerling, S.: Scippa: System-centric IPC provenance on android. In: Proceedings of the 30th Annual Computer Security Applications Conference, pp. 36–45 (2014)
8. Baiqin, W.: Binder Architecture and Core Components (2021). https://medium.com/swlh/binder-architecture-and-core-components-38089933bba
9. Benigni, D., Francomacaro, S.: INCITS 499-201x (revision of INCITS 499-2013), information technology -next generation access control -functional architecture (NGAC-FA) due date: The public review is from (2016). https://standards.incits.org/higherlogic/ws/public/download/85867/eb-2016-00808-002-Public-review-register-INCITS-499-Comments-due-2-28-2017.zip
10. Bezawada, B., Haefner, K., Ray, I.: Securing home IoT environments with attribute-based access control. In: Proceedings of the Third ACM Workshop on Attribute-Based Access Control, pp. 43–53 (2018)
11. Casey, P., Baggili, I., Yarramreddy, A.: Immersive virtual reality attacks and the human joystick. IEEE Trans. Dependable Secure Comput. **18**(2), 550–562 (2019)
12. Cheng, K., Tian, J.F., Kohno, T., Roesner, F.: Exploring user reactions and mental models towards perceptual manipulation attacks in mixed reality. In: Proceedings of the 32nd USENIX Security Symposium (2023)
13. De Giorgi, M.: System calls monitoring in android: an approach to detect debuggers, anomalies and privacy issues (2023)
14. Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., Wallach, D.S.: Quire: lightweight provenance for smart phone operating systems. In: USENIX Security Symposium, vol. 31, p. 3. San Francisco, CA (2011)
15. Falebita, O.S.: Secure web-based student information management system. arXiv preprint arXiv:2211.00072 (2022)
16. Ferraiolo, D., Chandramouli, R., Kuhn, R., Hu, V.: Extensible access control markup language (XACML) and next generation access control (NGAC). In: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control, pp. 13–24 (2016)
17. Ferraiolo, D., Gavrila, S., Jansen, W.: Archived nist technical series publication archived publication series/number: Title: Publication date(s): Withdrawal date: Superseding publication(s) (2014). https://doi.org/10.6028/NIST.IR.7987r1

18. George, C., Khamis, M., Buschek, D., Hussmann, H.: Investigating the third dimension for authentication in immersive virtual reality and in the real world. In: 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pp. 277–285. IEEE (2019)
19. George, C., et al.: Seamless and secure VR: adapting and evaluating established authentication systems for virtual reality. In: Proceedings of the CHI Conference on Human Factors in Computing Systems. NDSS (2017)
20. Google, D.: Application Sandbox (2024). https://source.android.com/docs/security/app-sandbox#:~:text=Because%20the%20Application%20Sandbox%20is,run%20within%20the%20Application%20Sandbox
21. Institute, A.N.S.: Information technology - next generation access control (NGAC). Information technology, ANSI, New York, NY (2020). Accessed 2023
22. Kaladharan, Y., Mateti, P., Jevitha, K.P.: An encryption technique to thwart android binder exploits. In: Berretti, S., Thampi, S.M., Dasgupta, S. (eds.) Intelligent Systems Technologies and Applications. AISC, vol. 385, pp. 13–21. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-23258-4_2
23. Kashmar, N., Adda, M., Ibrahim, H.: Access control metamodels: review, critical analysis, and research issues. J. Ubiquitous Syst. Pervasive Netw. **16**(2), 93–102 (2022)
24. Klepp, T.: Cruel intentions: enhancing androids intent firewall. Ph.D. thesis, Technische Universität Wien (2020)
25. Kozen, D.C., Kozen, D.C.: Depth-first and breadth-first search. In: The Design and Analysis of Algorithms. Texts and Monographs in Computer Science, pp. 19–24. Springer, New York (1992). https://doi.org/10.1007/978-1-4612-4400-4_4
26. Kraunelis, J., Fu, X., Yu, W., Zhao, W.: A framework for detecting and countering android UI attacks via inspection of IPC traffic. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2018)
27. Lawal, S., Krishnan, R.: Utilizing policy machine for attribute-based access control in permissioned blockchain. In: 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), pp. 1–6. IEEE (2021)
28. Lebeck, K., Ruth, K., Kohno, T., Roesner, F.: Securing augmented reality output. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 320–337. IEEE (2017)
29. Luo, S., Nguyen, A., Song, C., Lin, F., Xu, W., Yan, Z.: Oculock: exploring human visual system for authentication in virtual reality head-mounted display. In: 2020 Network and Distributed System Security Symposium (NDSS) (2020)
30. Lyvas, C., Lambrinoudakis, C., Geneiatakis, D.: Intentauth: securing android's intent-based inter-process communication. Int. J. Inf. Secur. **21**(5), 973–982 (2022)
31. Mathis, F., Williamson, J., Vaniea, K., Khamis, M.: Rubikauth: fast and secure authentication in virtual reality. In: Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1–9 (2020)
32. Odeleye, B., Loukas, G., Heartfield, R., Spyridonis, F.: Detecting framerate-oriented cyber attacks on user experience in virtual reality. In: VR4Sec: 1st International Workshop on Security for XR and XR for Security: Proceedings, pp. 1–5. Vancouver, B.C., Canada (virtual) (2021)
33. Raymer, E., MacDermott, Á., Akinbi, A.: Virtual reality forensics: Forensic analysis of meta quest 2. Forensic Sci. Int. Digital Invest. **47**, 301658 (2023)
34. Smalley, S., Craig, R.: Security enhanced (se) android: bringing flexible mac to android. In: Ndss, vol. 310, pp. 20–38 (2013)
35. Tseng, W.J., et al.: The dark side of perceptual manipulations in virtual reality. In: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, pp. 1–15 (2022)

36. Vondráček, M., Baggili, I., Casey, P., Mekni, M.: Rise of the metaverse's immersive virtual reality malware and the man-in-the-room attack & defenses. Comput. Secur. **127**, 102923 (2023)
37. Wei, Y.G., Lu, Y., Hu, X.Y., Sun, B.: Research and application of access control technique in 3D virtual reality system opensim. In: 2013 Sixth International Symposium on Computational Intelligence and Design, vol. 2, pp. 65–68. IEEE (2013)
38. Wu, D., Cheng, Y., Gao, D., Li, Y., Deng, R.H.: Sclib: a practical and lightweight defense against component hijacking in android applications. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY 2018), pp. 299–306. ACM, Tempe, AZ, USA (2018). https://doi.org/10.1145/3176258.3176336
39. Yang, Z., Li, C.Y., Bhalla, A., Zhao, B.Y., Zheng, H.: Inception attacks: immersive hijacking in virtual reality systems. arXiv preprint arXiv:2403.05721 (2024)
40. Yarramreddy, A., Gromkowski, P., Baggili, I.: Forensic analysis of immersive virtual reality social applications. In: A Primary Accout, 2018 IEEE Security and Privacy Workshops, pp. 186–196. IEEE. San Francisco (2018)