An Empirical Study on Oculus Virtual Reality Applications: Security and Privacy Perspectives

Hanyang Guo
Hong Kong Baptist University
Hong Kong, China
Sun Yat-sen University
Zhuhai, China
guohy36@mail2.sysu.edu.cn

Hong-Ning Dai*
Hong Kong Baptist University
Hong Kong, China
hndai@ieee.org

Xiapu Luo
The Hong Kong Polytechnic University
Hong Kong, China
csxluo@comp.polyu.edu.hk

Zibin Zheng
Sun Yat-sen University
Zhuhai, China
zhzibin@mail.sysu.edu.cn

Gengyang Xu
Hong Kong Baptist University
Hong Kong, China
21253277@life.hkbu.edu.hk

Fengliang He
Hong Kong Baptist University
Hong Kong, China
csflhe@comp.hkbu.edu.hk

ABSTRACT

Although Virtual Reality (VR) has accelerated its prevalent adoption in emerging metaverse applications, it is not a fundamentally new technology. On one hand, most VR operating systems (OS) are based on off-the-shelf mobile OS (e.g., Android). As a result, VR apps also inherit privacy and security deficiencies from conventional mobile apps. On the other hand, in contrast to conventional mobile apps, VR apps can achieve immersive experience via diverse VR devices, such as head-mounted displays, body sensors, and controllers though achieving this requires the extensive collection of privacy-sensitive human biometrics (e.g., hand-tracking and face-tracking data). Moreover, VR apps have been typically implemented by 3D gaming engines (e.g., Unity), which also contain intrinsic security vulnerabilities. Inappropriate use of these technologies may incur privacy leaks and security vulnerabilities although these issues have not received significant attention compared to the proliferation of diverse VR apps. In this paper, we develop a security and privacy assessment tool, namely the VR-SP detector for VR apps. The VR-SP detector has integrated program static analysis tools and privacy-policy analysis methods. Using the VR-SP detector, we conduct a comprehensive empirical study on 500 popular VR apps. We obtain the original apps from the popular Oculus and SideQuest app stores and extract APK files via the Meta Oculus Quest 2 device. We evaluate security vulnerabilities and privacy data leaks of these VR apps by VR app analysis, taint analysis, and privacy-policy analysis. We find that a number of security vulnerabilities and privacy leaks widely exist in VR apps. Moreover, our results also reveal conflicting representations in the privacy policies of these apps and inconsistencies of the actual data collection with the privacy-policy statements of the apps. Based on these findings, we make suggestions for the future development of VR apps.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

KEYWORDS

Virtual Reality, Metaverse, Static Analysis, Security and Privacy

ACM Reference Format:
Hanyang Guo, Hong-Ning Dai*, Xiapu Luo, Zibin Zheng, Gengyang Xu, and Fengliang He. 2024. An Empirical Study on Oculus Virtual Reality Applications: Security and Privacy Perspectives. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639082

1 INTRODUCTION

Virtual Reality (VR) [57] has recently received a boosted development. Diverse VR devices and VR systems have been developed by Meta (previously Facebook), Apple, Microsoft, ByteDance, Sony, HTC, etc. As reported by Fortune [27], the global VR market size is projected to grow from $25.11 billion in 2023 to $165.91 billion by 2030. The proliferation of VR devices and VR systems has also greatly driven the development of the metaverse, which emphasizes users' immersive experience in virtual worlds [37] and the real-time interactions with 3D models in a VR environment.

Despite its rapid development, VR is not a fundamentally novel technology. VR's conceptual prototypes were established several decades ago by implementing a computer simulation system to generating 3D objects in virtual worlds. The recent development of hardware and software has fastened the adoption of VR technology. With the proliferation of VR devices and metaverse systems, a large number of VR apps have been developed and released. Most of these VR apps are running on top of off-the-shelf mobile operating systems (OS), such as Android (as well as its variants), Sony Orbis OS (originated from FreeBSD 9), and Apple visionOS (based on iOS). As a result, VR apps share common features with conventional mobile apps and also inherit their intrinsic deficiencies. For example, many VR apps run on top of Android OS with underlying VR devices (e.g., Meta's Oculus Quest 2 [13] and ByteDance's Pico 4

* Corresponding author.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3639082

[25]). Consequently, VR apps developed based on these VR devices are packaged as Android PacKage (APK) files. After decompiling APK files, these VR apps also generate similar files to Android apps, such as `AndroidManifest.xml`, resource files, java files, and so on.

Despite the similarity to Android mobile apps, VR apps have unique characteristics different from conventional mobile apps. (i) VR apps include not only Personally identifiable information (PII) data like Android apps, but also VR-specific PII data (e.g., VR device ID, Controller ID, and sensor ID). (ii) Many VR apps have been developed on top of specific VR platforms (e.g., Oculus Quest SDK) and 3D game engines (e.g., Unity, Unreal Engine, etc.) for achieving an immersive user experience in the 3D environment. (iii) VR apps request not only identity information data like Android mobile apps, but also extensive access to human biometrics, such as iris or retina scans, fingerprints, hand-tracking as well as face-tracking data, and voice. The use of new data types requires additional permission authentication, which can place new requests on the specification of configuration files, e.g., including new VR-related permission flags in `AndroidManifest.xml` in addition to conventional Android permission flags. (iv) VR apps have more descriptions of access norms for human biometrics in their privacy policies, sharply different from conventional Android apps.

Sharing common features of Android apps while possessing different features, VR apps are exposed to not only known security and privacy vulnerabilities inherited from Android apps but also emerging security and privacy risks. Unfortunately, a comprehensive study on VR apps is largely missing in the literature compared to the proliferation of VR apps [23]. Ignorance of these emerging security risks will greatly dampen users' enthusiasm for using VR apps and purchasing VR devices as well as VR services [36, 56]. For example, as discovered in Bigscreen (i.e., a famous virtual social VR app), a vulnerability allows strangers to perform various intrusive activities without the user's consent, such as turning on the user's microphone and eavesdropping on private conversations [10].

In order to address the above issues, we propose a VR Security and Privacy assessment tool for VR apps, called *VR-SP Detector* for detecting potential security vulnerabilities and privacy risks. To the best of our knowledge, this is the first study on evaluating security and privacy issues of VR apps from code analysis and privacy-policy analysis. Despite a recent study (OVRSeen) on analyzing privacy policies of VR apps [53], it mainly focuses on network traffic rather than the implemented codes of VR apps. Our VR-SP detector integrates both static code analysis tools and privacy-policy analysis techniques, thereby effectively revealing security and privacy risks. Based on the VR-SP detector, we conduct a comprehensive empirical study on 500 popular VR apps (i.e., more than 3 × of OVRSeen) from the largest VR app store SideQuest.

After extracting the original APK file for each app after installing it in Meta Oculus Quest 2, we then decompile the APK to get not only the configuration file but also the source codes as well as the intermediate representation (IR) files. We next conduct a comprehensive analysis of the decompiled codes of each app. In particular, we conduct an analysis on the configuration file `AndroidManifest.xml` of each app to get app basic information and permission usage information. We adopt pattern recognition techniques to detect Android-related general security and privacy vulnerabilities from Java files and the corresponding IR (i.e., Smali) files. We also use a taint analysis framework to detect private data leaks. Since most VR apps utilize Unity to achieve immersive environment rendering, biometric data capture, and even In-App Purchasing (IAP) service [15, 64], we hence leverage a static binary analysis framework to detect the flows, biometric data usage, and IAP data in Unity-developed VR apps. At last, we adopt a privacy policy analysis based on natural language processing (NLP) to detect inconsistencies in the privacy policies of VR apps. We have obtained many insightful findings, on which, we provide some suggestions on the development of VR apps. In summary, the main contributions of this work are summarized as follows.

- We propose an automatic security and privacy evaluation tool for metaverse-related VR apps. This tool can detect not only general vulnerabilities but also sensitive (PII and biometric) data leaks.
- Our tool integrates static analysis and privacy-policy analysis technologies with consideration of the unique characteristics of security and privacy issues of VR apps. Specially, we detect security threats of VR apps by pattern matching and taint analysis from the decompiled code and use an NLP-based method to analyze privacy policies.
- We run our tool on 500 popular VR apps and find that more than 95.40% of the apps exist no root detection (No RD) vulnerability and 37.20% exist insecure random generator (IRG) vulnerability. Moreover, 44.40% of the apps invoke functions using biometric data though there are no requests in the manifest file. Moreover, 55.20% of apps have no privacy policy and 11.00% of apps have contradictory statements in privacy policies.
- Based on the findings, we provide some advice on VR app development. We make our tool available at https://github.com/Henrykwokkk/Meta-detector.

2 BACKGROUND

2.1 Taxonomy of VR apps

VR apps in the metaverse context have different characteristics from conventional VR apps. Conventional VR apps usually create virtual content for a single user whose activities typically occur alone, e.g., simulating a single-user adventure, playing a single-player game, and watching VR movies alone. Differently, VR apps in the metaverse context emphasize the interaction among multiple users. Moreover, this kind of VR app typically creates virtual elements from real-world elements such as buildings, objects, and characters. Further, they also greatly extend virtual spaces from computer games to education, socialization, and online business activities. Considering news reports and research papers [12, 26, 30, 41], we mainly consider the following five types of VR apps:

- **Virtual society** [12, 41]. These VR apps provide users with virtual spaces to interact and socialize with others. Typical apps include Rec Room, VRChat, etc.
- **Gaming** [26, 41]. A large body of VR apps are themed with games, such as Pavlov VR, Echo VR, etc.
- **Art and culture** [30]. Many VR apps support virtual museums, exhibitions, concerts, and other cultural and artistic events, such as Forbidden City Journey, Gravity Sketch, etc.
- **Education** [41]. As a growing trend in the metaverse, VR apps can provide virtual classrooms, learning resources, and online courses, such as EngageVR, VR Anatomy, etc.

- **Business and finance** [38]. Many VR apps provide virtual stores, trading platforms, and financial services, e.g., Decentraland.

2.2 Security and Privacy Vulnerabilities of VR Apps

2.2.1 OS-related Security and Privacy Vulnerabilities. Since a large body of VR devices run on top of Android OS or its variants, VR apps share some similar vulnerabilities with Android apps running on top of mobile devices like mobile phones although VR devices also have different features from mobile phones. We investigate the following security and privacy vulnerabilities, which are bestowed on new metaverse/VR features though they originated from Android security analysis [35, 50].

Insecure Flag Settings: These vulnerabilities come from insecure flags in the configuration file `AndroidManifest.xml`, such as `allowBackup`, `debuggable`, and `clearTextTrafiic`.

Dangerous Permission Usage: These vulnerabilities are related to the misuse of dangerous Android permission requests, such as `location`, `camera`, `microphone`, etc. These vulnerabilities also exist in VR apps. Differently, we also consider permissions requests from other peripheral VR-related devices, e.g., controllers.

PII Data Leaks: Some of these vulnerabilities are related to PII data (e.g., user, name, password, email, and phone) leaks.

General Vulnerabilities: State-of-the-practice tools [16, 29, 46, 50] also presented general vulnerabilities. We summarize them into the following nine categories in the VR context.

(1) *SQL Database Injection (SDI) in VR apps* [2] means that an attacker can insert additional SQL statements to the end of a predefined query statement or input in an application to trick the database into executing an un-authorized query. (2) *Insecure certificate validation (ICV) in VR app client* [40] means that it might allow an attacker to spoof a trusted entity by interfering in the communication path between the host and client if a certificate is invalid or malicious. (3) *Insecure random generator (IRG)* [24] means some insecure random number methods that can produce predictable values (e.g., virtual room passwords) as a source of randomness in a security-sensitive context. (4) *Insecure Webview Implementation (IWI)* [45] means that the VR app allows loading HTML contents and HTML pages within the application. (5) *IP Disclosure (IPD)* [32] is a vulnerability that can be exploited by an attacker to obtain internal information from VR Apps' IP addresses. (6) *Remote Webview Debugging (RWD) of VR apps* [5] means to enable webview debugging in VR apps. (7) *Unsafe sensitive data (such as user input by the virtual keyboard) cryptographic algorithms* include improper encrypt functions (IEF) [48] and insecure hash functions (IHF) [61]. (8) *Root Detection (RD) of VR devices* [62] means to detect the function usage that requires root access and check if the app asks to detect the rooted device. (9) *VR-related Tackers* [28] include not only trackers in general Android apps (e.g., *Google Firebase Analytics*) but also other VR-specified trackers (e.g., *Unity3d Ads*).

how does IRG work?

2.2.2 VR-platform-related Security and Privacy Vulnerabilities. To achieve an immersive experience in the metaverse, VR apps typically implement diverse VR features, such as avatar modeling, 3D rendering, and 3D interaction. VR development frameworks, such as Unity [52] and Unreal Engine (UE) [20] have been increasingly adopted. Both Unity and UE have been implemented in C++ though Unity has been partially implemented in C#. Moreover, to capture users' location and movement, human biometrics, such as hand tracking, eye movement, face tracking, and body tracking have been collected and analyzed. VR device manufacturers (e.g., Meta Oculus, HTC, Bytedance) provide developers with SDKs to achieve immersive VR features. However, the adoption of VR development frameworks and VR device SDKs inevitably causes new security vulnerabilities and privacy risks; this feature is *sharply different from the development of conventional Android Apps*. In particular, we categorize security and privacy vulnerabilities related to VR development frameworks and VR device SDKs as VR-platform-related vulnerabilities, which are summarized as follows.

Table 1: Keywords of Five VR App Categories

Categories	Keywords
Virtual Society	Social VR, Social Media, Virtual/Social Communications
Gaming	VR Games, VR Entertainment, Metaverse Games
Art and Culture	Culture, Art, Museum
Education	Education, Teaching, Learning
Business and Finance	Finance, Business, Economic, NFT, DeFi

- **New Permission Requests**: The use of new data (such as human biometrics) may introduce the problem of managing new permissions. Although device manufacturers provide new `<uses-permission>` tags for these new permission requests in the `AndroidManifest.xml` file, the technical regulation of using these permissions is still worth investigating.
- **Misuse of Human Biometrics**: Unity or UE-based frameworks include 3D rendering and exploit human biometrics by data collection APIs. For example, Oculus Unity SDKs provide some functions of collecting hand-tracking, eye-tracking, body-tracking, and face-tracking data[1]. The misuse/abuse of such sensitive may cause severe security and privacy issues.
- **In-App Purchasing (IAP) Vulnerabilities**: With the prevalent adoption of IAP services in VR apps, it also incurs security risks. For example, players may purchase virtual items, such as virtual avatars, virtual currency, or NFT assets. Inappropriate authentication or authorization when purchasing virtual items may be the root cause of security vulnerabilities.
- **VR-specific PII Data Leak**: VR apps may suffer from the leakage risks of sensitive PII data, which include not only traditional PII data from Android systems but also VR-relevant PII data from VR devices (HMDs and peripheral devices).
- **Privacy Policy Weakness**: Recently, several studies have been conducted to analyze the privacy policies of mobile apps (e.g., Android apps) to identify problems and verify their reliability [8, 34, 54], despite few studies on privacy policy analysis on VR apps. As newly emerging applications, VR apps are undergoing privacy policy weakness caused by both incomplete technical regulation originating from traditional Android development norms and new issues of VR devices in using and collecting users' privacy-sensitive data.

[1]We mainly consider these functions: `OVRHand.OVRMesh`, `IOVRMeshDataProvider.GetMeshType`, `OVRBody.OVRSkeletonRenderer.IOVRSkeletonRendererDataProvider.GetSkeletonRendererData`, `VREyeGaze.CalculateEyeRotation`, and `VRFaceExpressions.ToArray`, etc.

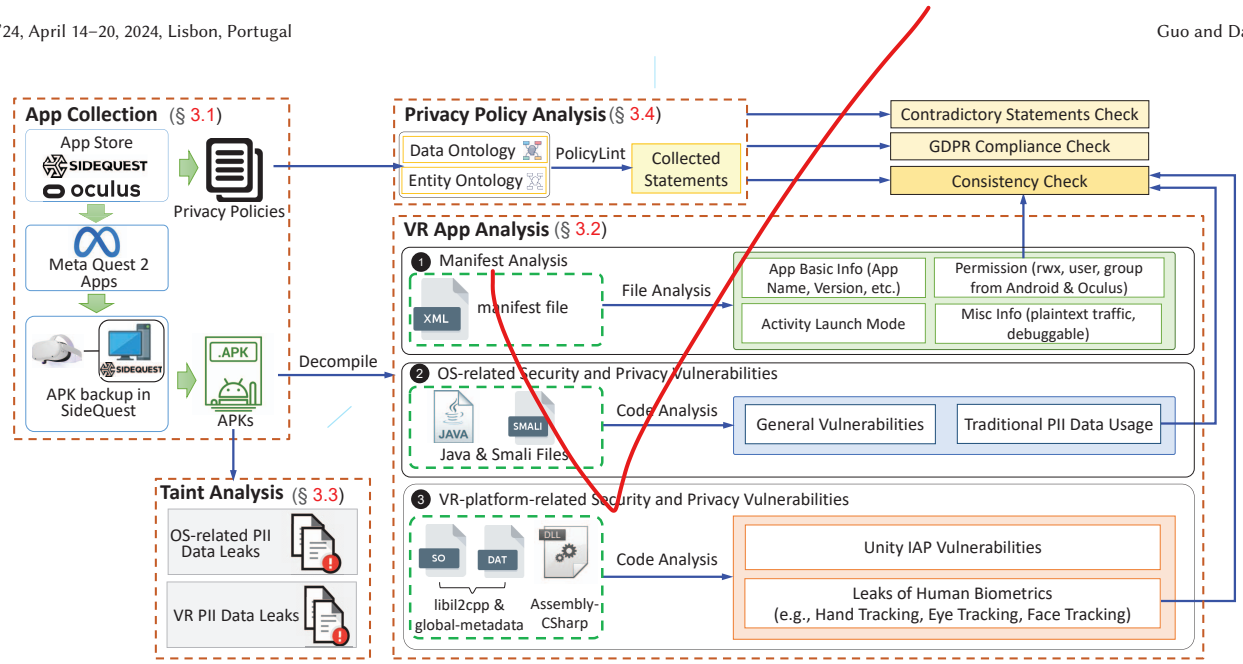


Figure 1: The Overview of *VR-SP Detector*

3 METHODOLOGY OF VR-SP DETECTOR

This section elaborates on the proposed VR-SP detector to analyze Oculus VR apps. Figure 1 depicts the overall framework of the VR-SP detector. The proposed VR-SP detector works in the following steps: (1) App Collection, (2) VR App Analysis, (3) Taint Analysis, and (4) Privacy Policy Analysis, which are described as follows.

3.1 App Collection

According to rankings and popularity, we collect the 500 most popular VR apps from both the official Oculus store and SideQuest. The latter is the most popular third-party store endorsed by Meta. Compared with the Oculus store, SideQuest receives a larger popularity and contains more apps. To obtain the original app files (not those from unauthorized parties), we download these apps from either the Oculus store or SideQuest. We then install them on Meta Quest 2 (aka Oculus Quest 2), one of the most popular VR devices. It is worth mentioning that our static analysis of these VR apps is not affected by VR devices even though some full-fledged features (e.g., eye-tracking and face-tracking) may require to be executed on higher-end VR devices (e.g., Oculus Meta Quest Pro). These apps are collected according to the categories specified in § 2.1. We obtained VR apps by the popularity and the keywords specified in Table 1. We use the "hot" ranking in SideQuest as the popularity metric and consider only those free apps. At last, we collected 500 apps including 81 virtual social apps, 149 game apps, 115 art and culture apps, 115 education apps, and 40 business and finance apps (the distribution of those apps is given in § 4).

but how about not free apps?

After collecting apps, we then extract APK files from those apps. Since neither the Oculus official store nor SideQuest store provides a direct downloading link for APK files, we extract APK files by installing each app on the Oculus Quest 2 device. We first connect Oculus Quest 2 to the PC via a USB cable and install the app into Oculus Quest 2 via SideQuest. After that, we exploit the "APK backup" function of SideQuest to extract the APK file corresponding to each app. For further privacy policy analysis (in § 3.4), we also collect the policy statement of each of those 500 apps.

3.2 VR App Analysis

We conduct VR app analysis based on the APK file extracted from the installed app. Since we cannot directly analyze a VR app, we first decompile its APK file. In order to attain a detailed program structure and development code information, we adopt Androguard [17], an open-source Python tool capable of extracting different kinds of information from the individual components of an APK file [59] to obtain configuration files, Dalvik bytecode, and source code. Figure 2 shows an example of the APK structure of VR apps.

From the decompiled files, we conduct ❶ Manifest Analysis, detect ❷ OS-related Security and Privacy Vulnerabilities, and identify ❸ VR-platform-related Security and Privacy Vulnerabilities on the decompiled code of each app as shown in Figure 1. The detailed analysis procedure is elaborated as follows.

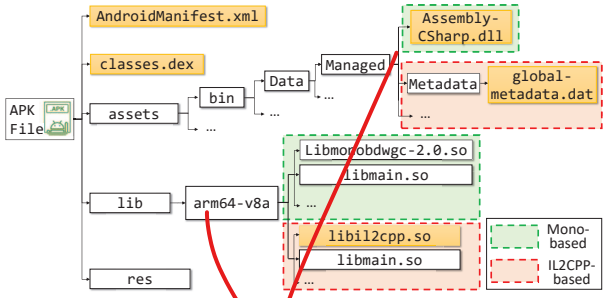


Figure 2: APK Structure of VR App

Table 2: Pre-defined Rules of OS-related Vulnerabilities

Vulnerability Types	Pre-defined Rules
SDI	Search for dangerous SQL query method signatures in Smali files.
ICV	Search for SSL or host verifier method signatures in Smali files.
IRG	Search for random number generator method signatures in Smali files.
IWI	Search for webview client and SSL errors received enable related method signatures in Smali files.
RWD	Search for webview-debug-related method signatures in Smali files.
IEF	Search for cryptographic class and instantiation method signature, and match insecure encryption keywords in Smali files.
IHF	Search for message digest class, instantiation method signature, and match weak hash pattern in Smali files.
RD	Search for root and sudo access string.
IPD	Search IPV4, IPV6 and private address strings by regular expression matching.
Trackers	Find the tracker code signature that is included in the tracker list in the Smali files.

3.2.1 Manifest Analysis. After decompilation, we extract `AndroidManifest.xml` configuration file and parse it to extract key features of each VR app. In particular, we collect four types of features: 1) app basic information including `app_name`, `package_name`, `version_code` and `sdk_version` of the app; 2) permission information containing the permission requests, which include not only predefined permissions by Android but also those newly defined by VR devices (e.g., Oculus Quest 2); 3) activity launch mode, which refers to the start mode of activity in the task stack; and 4) miscellaneous information including `allow_backup` and `use_plaintext_traffic` flag information; the settings of these flags have an impact on the security of data transmission of VR apps.

Regarding permission information, there are nine predefined permissions from Oculus (i.e., `com.oculus.permission`): `HAND_TRACKING`, `RENDER_MODEL`, `TRACKED_KEYBOARD`, `USE_ANCHOR_API`, `FACE_TRACKING`, `TOUCH_CONTROLLER_PRO`, `BODY_TRACKING`, `EYE_TRACKING`, and `DEVICE_CONFIG_PUSH_TO_CLIENT`. Moreover, the permission protection level can be divided into four categories: *normal*, *dangerous*, *signature*, and *signatureOrSystem* according to Android Development Documentation. We classify all the permissions defined by Oculus as *dangerous* protection levels because they are all related to requests for sensitive information. With respect to activity launch mode, it can be used to detect the task-hijacking vulnerability in VR apps. Specifically, we identify the launch mode of activities that is `singleTask` without setting `taskAffinity` label, because this kind of activity may cause the task-hijacking vulnerability (i.e., StrandHogg) [44].

3.2.2 Detecting OS-related Security and Privacy Vulnerabilities. We extract the decompiled partial Java files (without detailed variable names and method signatures) and Smali files that include Dalvik bytecode from `classes.dex` to detect some general security and privacy vulnerabilities indicated in § 2.2.1 in VR apps. Specifically, we use pre-defined patterns based on [50] to detect potentially vulnerable methods (functions) and strings from Java codes or Dalvik bytecodes. We then search whether the function call paths exist. If the call paths or strings exist, we consider that the app being evaluated has this type of vulnerability. For example, suppose an app includes *Cipher* and *AES/ECB* keywords in the app. In that case, it may have IEF because the app uses the Electronic Code Book (ECB) mode in the cryptographic encryption algorithm. Since the same block of plaintext is encrypted into the same block of ciphertext in ECB, it may cause the leakage of encrypted messages [19]. In summary, there are nine types of security and privacy vulnerabilities that we detect in VR apps. Table 2 summarizes pre-defined patterns. Moreover, we also extract the methods (functions) that collect and use PII data. We search method signatures by using PII keywords, such as *user*, *password*, *username*, *phone*, *id*, and *email* to identify PII data usage, which can be used for checking privacy policy consistency (to be depicted in § 3.4).

3.2.3 Identifying VR-platform-related Security and Privacy Vulnerabilities. This module focuses on detecting VR-platform-related security and privacy vulnerabilities, such as Unity IAP vulnerabilities and human biometrics leaks. As mentioned in § 2.2.2, VR apps adopt game engines to achieve immersive features. We mainly consider Unity, which is the most dominating framework in VR software development [39]. We adopt a static native binary analysis tool [64] to detect IAP vulnerabilities in VR apps developed based on Unity. There are two ways to run C# programs on Unity. One is to compile the C# code to .NET Common Intermediate Language (CIL) and use a Mono Virtual Machine (VM) to execute the CIL code at run time. This manner is called Mono-based (see green dash box in Figure 2). The other is to further transfer CIL codes into C++ codes and then compile C++ codes into native binaries. This method is IL2CPP-based (see red dash box in Figure 2). Corresponding to Mono-based and IL2CPP-based methods, APK files are named Mono-based and IL2CPP-based apps, respectively. Specifically, as for Mono-based apps, we extract the compiled logic code file `Assembly-CSharp.dll` and use a reversed tool called *dnSpy* [1] to get the C# source code. As for IL2CPP-based apps, we extract the compiled Unity binary file `libil2cpp.so` and the function mapping file `global-metadata.dat` from the decompiled APK file. Different from previous work [64], which only analyzed the IL2CPP-based apps, our proposed VR-SP detector focuses on both types of apps. Specifically, we use taint analysis to track the payment receipt data with the use of both the binary file and the mapping file. We define the method `UnityEngine.Purchasing.Product.get_receipt` as the taint *source* and the return value as tainted data. Regarding the detection of local-verification vulnerabilities (i.e., validating transactions only on the local server rather than asking the app store to verify the transaction), we define the method `CrossPlatformValidator.Validate` as *sink*. If the tainted data is received while does not reach the network API, a *local-verification vulnerability* is considered as detected. If the payment data is not sent externally (e.g., via a network API) and there is no local verification API involved, then a *no-verification vulnerability* is considered as detected.

Different from traditional Android apps, VR apps may collect human biometrics, such as eye location, hand coordinates, and

so on. To tackle this emerging issue, we also trace the propagation of human biometrics in the proposed framework. Specifically, we extract the functions of collecting hand-tracking, eye-tracking, body-tracking, and face-tracking data described in § 2.2.2 according to the Oculus development document. We also conduct a taint analysis on those functions. Thereafter, we detect the usage of these biometric functions to check the risk of data leaks and also check whether their use is consistent with the permission request and the privacy policy statement (in § 3.4).

3.3 PII Data Leaks Identification

Since VR apps have high risks of PII data leaks, we conduct a taint analysis to detect sensitive information leaks based on data flow analysis. Firstly, we define VR-related PII data, which includes not only OS-related PII data (e.g., *username*, *phone*, *email*, etc.), but also VR-specific PII data (e.g., *VR device ID*, *Controller ID*). Thereafter, referring to [7], we define the methods that transmit sensitive data as *sources* (e.g., `getString(int)`, `getIntent()`, etc.). Moreover, we consider methods that may get sensitive data without user input, such as `getAddress()` for address information acquisition or `database.Cursor.getAllVisitedUrls()` for URLs queries in the database as *sources*, too. The return values of the above methods are the tainted data [7]. These tainted data may not be directly accessed with authentication but they may cause leaks if these data flow to a method can be accessed by unauthorized users. We define these modes as *sinks*, such as `sendBroadcast()` and `sendDataMessage()`.

After defining the *source*, *sink*, and sensitive data, we search the lifecycle and method callback in the activity to construct the control flow graph (CFG). Based on CFG, we detect each defined source and taint the sensitive data. Then, we execute the data flow analysis to track the tainted data. If the tainted data flows from a source to a sink, we label it as a potential privacy-leak path. After conducting a backward flow analysis, which aims to confirm the vulnerable code is reachable (i.e., not dead code) to reduce false positives, we identify that the app has a risk of privacy leak. For instance, if the return value of `getAddress()` flows to a `sink`, such as `sendDataMessage()` and any user or app that can access it without permission, we identify a privacy leak existing.

3.4 Privacy Policy Analysis

The privacy policy is a complete and clear description of the practices of product and service providers in collecting, storing, using, and providing personal information to the public [11]. As mentioned in § 2.2.2, few studies focus on privacy policy analysis on VR apps. In our VR-SP detector, we implement a privacy policy analysis module to check the contradictory statements and the consistency between the privacy policy and the app analysis results. We use PolicyLint [3], which transfers each statement in the privacy policy to plain texts and takes them as input. The output of the tools is collected statements formatted like *<entity, action, data type>*, where *entity* refers to the app or third-party platform that receives the privacy data, *action* specifies the manner in which entities process data and *data type* is the type of privacy data. The categories of entity and data type are defined in [53]. For example, the privacy policy sentence "*We will collect your photo information and voice information for AI face pinching*" can be input to PolicyLint to generate the collection statements *<we, collect, voice information>* and *<we, collect, photo information>*. We check privacy policies to see whether there are different collection operations for the same data type, i.e., both collection and non-collection.

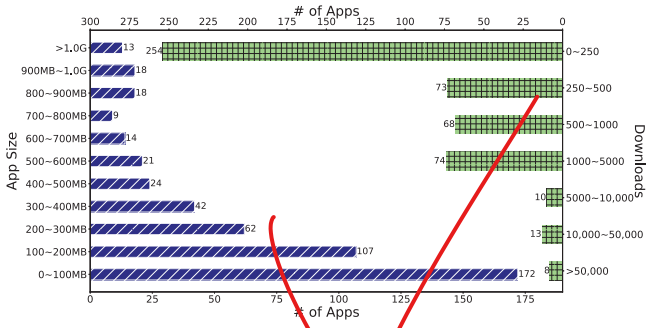


Figure 3: App size & Downloads Distribution

We also detect whether each privacy policy complies with General Data Protection Regulation (GDPR). GDPR was enacted by the European Union in 2018 and is one of the most well-known data privacy protection laws in the world [33]. We adopt a free online service called GDPRWise [9] to conduct GDPR compliance checks.

The consistency check module in our VR-SP detector includes three components. Firstly, we use the collected permission feature in the *AndroidManifest.xml* file and adopt the permission request information in manifest analysis to make a consistency check for confirming the reliability of privacy policies. We extract the apps that request permissions `HAND_TRACKING`, `FACE_TRACKING`, `BODY_TRACKING` and `EYE_TRACKING`. We check whether their privacy policies have statements about the use of these sensitive data. We conduct a consistency check between the policy statements and the corresponding permission requests. Secondly, we use the result of decompiled Java and Smali code analysis. We extract methods that collect PII information by keyword searching and check whether their privacy policies have statements about the use of these PII data. Thirdly, we also conduct a consistency check between biometric collection function usage and privacy policy.

4 ANALYSIS AND RESULTS

In this section, we evaluate the proposed VR-SP detector by conducting comprehensive experiments on the collected 500 VR apps. We mainly consider the following four research questions (RQs).

RQ1: What is the manifest vulnerability profile of VR apps?
RQ2: What are VR apps' major OS-related security and privacy vulnerabilities?
RQ3: What are VR apps' major VR-platform-related security and privacy vulnerabilities?
RQ4: To what extent is PII data leaked?
RQ5: How do the VR app developers comply with the privacy policies?

Data Preparation. As mentioned in § 3.1, we collect 500 VR apps from five categories in the context of VR (the complete app list is given in our repository). Figure 3 summarizes the app size and download distribution of those 500 VR apps. It can be found that

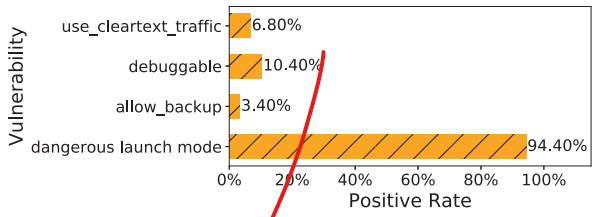


Figure 4: Manifest Analysis Result

many apps are less than 100MB, which may own to limited space in current VR devices. With respect to the number of downloads, most VR apps have been downloaded less than 5,000 times, implying that the development of VR apps is still in its early stage.

4.1 RQ1: What is the manifest vulnerability profile of VR apps?

Analyzing the manifest file of each app, we can find vulnerability risks, as shown in Figure 4. It can be found that 94.40% of the VR apps have activities having dangerous launch modes. For these activities, we further analyzed their specific contents and reported the results in Table 3. We find that most of the activities are `com.unity3d.player.UnityPlayerActivity` since these apps are developed based on Unity. Some other activities with dangerous launch modes are from third-party platforms (e.g., epicgame, google, etc.). Further, a root activity makes dangerous launch mode attributes be insecure since it is possible for other malware to read the contents of the calling intent. Table 4 shows examples of apps that have the most dangerous launch mode activities, where these apps are anonymized by MD5 encryption with the first five prefix letters. These apps need to be used with caution security risks.

Table 3: Dangerous Activities of VR apps

Types of Dangerous Activities	Amount
`com.unity3d.player.UnityPlayerActivity`	421
`com.epicgames.ue4.GameActivity`	32
`com.google` Domain Activity	12
`com.epicgames.unreal` Domain Activity	6
`com.deepinc` Domain Activity	2
`com.microsoft` Domain Activity	2
`com.oculus` Domain Activity	2
Others	13

Compared with *dangerous launch mode*, the percentage of apps containing other types of manifest vulnerabilities is relatively small. The positive rates of *allow_backup*, *debuggable* and *use_cleartext_tracffic* are 3.40%, 10.40% and 6.80%, respectively. Enabling *allow_backup* and *debuggable* causes a risk of coping and tampering with data from the device. This is even riskier in VR devices with human biometrics collected. Further, *use_cleartext_tracffic* may cause a Man-in-the-Middle (MITM) [14] attack.

We also count the most used dangerous permissions and the number of the used Oculus permissions mentioned in § 3.2. As reported in Figure 5, most VR apps use INTERNET permission though

Table 4: App Examples with Most Dangerous Launch Mode Activities

App MD5 Prefix	Activity
203fc~	com.google.firebase.auth.internal.GenericIdpActivity com.google.firebase.auth.internal.RecaptchaActivity com.unity3d.player.UnityPlayerActivity
d3458~	com.google.firebase.auth.internal.GenericIdpActivity com.google.firebase.auth.internal.RecaptchaActivity com.unity3d.player.UnityPlayerActivity
a0165~	com.deepinc.liquidcinemasdk.SettingsActivity com.deepinc.liquidcinemasdk.VideoSixGridActivity
6e800~	com.google.firebase.auth.internal.GenericIdpActivity com.google.firebase.auth.internal.RecaptchaActivity
0163c~	com.google.android.play.core.missingsplits.PlayCoreMissingSplitsActivity com.unity3d.player.UnityPlayerActivity
4e336~	com.pico.loginpaysdk.auth.TransferStationActivity com.storm.forbiddencityjourney.MainActivity
9bbd5~	com.pico.loginpaysdk.auth.TransferStationActivity com.unity3d.player.UnityPlayerActivity
a2114~	com.epicgames.ue4.GameActivity com.google.android.play.core.InstallActivity

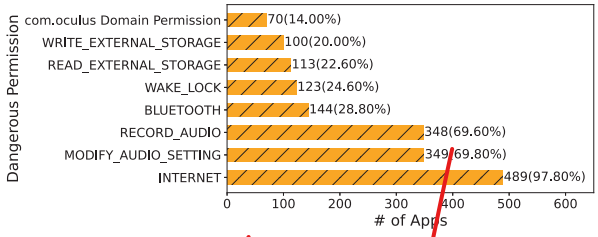


Figure 5: Dangerous Permission

some of them use permissions related to sound recording. This is because the social properties and immersive nature of VR apps require the Internet access and record the user's voice. Meanwhile, 70 apps use Oculus permissions to attain sensitive data. It is necessary to check whether dangerous permissions are secure before installing them. It is also important to manage app permissions by checking which permissions are allowed and declining if necessary.

Answer to RQ1: Most of the VR apps have a dangerous launch mode and sound-recording. Some apps have backup, debug, and network traffic misuse.

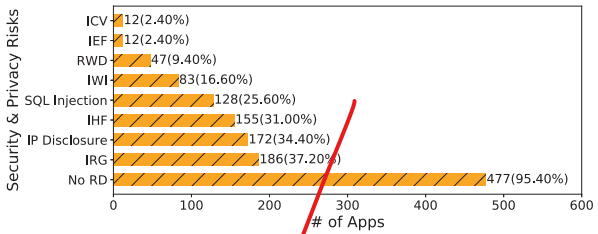


Figure 6: Results of Code Analysis

Table 5: App Examples with Security and Privacy Risks

App MD5 Prefix	cda01~	4dbd4~	0163c~	9bbd5~	58dea~	7d3df~	3aa85~	54208~
SDI	✓	✓	✓	✓	✓	✓	✓	✓
ICV	✗	✗	✗	✓	✗	✗	✗	✗
IRG	✓	✓	✓	✓	✓	✓	✓	✓
IWI	✓	✓	✓	✓	✓	✓	✓	✓
IPD	✓	✓	✓	✓	✓	✓	✓	✓
RWD	✗	✓	✓	✗	✗	✓	✓	✓
IEF	✓	✗	✗	✓	✓	✗	✗	✗
IHF	✓	✓	✓	✓	✓	✓	✓	✓
No RD	✓	✓	✓	✓	✓	✓	✓	✓

4.2 RQ2: What are VR apps' major OS-related security and privacy vulnerabilities?

As mentioned in § 3.2, we detect security and privacy vulnerabilities from decompiled Java and Smali files. The results are shown in Figure 6. It can be found that most VR apps have the no-root-detection vulnerability of VR devices (477 apps), thereby leading to private information leakage caused by accessing databases with root privileges. Meanwhile, 186 apps have insecure random generators, which may cause private information (e.g., location) to be tracked by speculating random numbers on critical functions such as `generateDefaultSessionId` `ChangeWatermarkPosition` used in virtual social and business apps. In addition, IHF can also cause a serious security issue in VR apps. Attackers can exploit IHFs to conjecture users' input via analyzing typing activities on virtual keyboard [58]. Table 5 shows examples of apps (anonymized by MD5) having the most security and privacy risks.

We also identify trackers used in VR apps by code analysis. Table 6 summarizes the most typical 15 trackers of VR apps. We observe that 108 VR apps use trackers, implying the prevalent usage of trackers in VR apps. Although the use of trackers can help VR app developers to provide users with customized services, it may expose users to the risk of privacy breaches. For example, [4] indicated that sharing sensitive data with distinct advertisers (trackers), such as *Unity3d Ads* is egregious.

Table 6: Tracker Analysis

Trackers	# of Apps
Google Play Billing Library / Service	62
Unity3d Ads	40
Google Firebase Analytics	15
GameAnalytics	8
Umeng Analytics	3
Umeng Common SDK logging	3
Bugsnag	3
Amplitude	2
Microsoft Visual Studio App Center Analytics	2
Microsoft Visual Studio App Center Crashes	2
Singular	1
AppMetrica	1
Branch	1
Bugfender	1
Google AdMob	1

Table 7: Inconsistency of Biometric Function Usage

Biometric Function	# of Apps
Hand-Tracking Function	209
Eye-Tracking Function	59
Body-Tracking Function	63
Face-Tracking Function	62

Answer to RQ2: Most VR apps have the no-root-detection vulnerability of VR devices. A significant number of apps have used trackers.

4.3 RQ3: What are VR apps' major VR-platform security and privacy vulnerabilities?

We next detect Unity IAP vulnerabilities and the usage of human biometrics. Regarding CIL to C++ Unity-based apps, we further obtain 332 apps from 500 apps. Depending on different approaches (Mono-based and IL2CPP-based), we obtain 124 Mono-based apps. We taint the Unity IAP function and biometric function. As for the results of Unity-based code analysis (i.e., C# code analysis), we find that there are 28 apps adopting the Unity IAP function. According to the taint analysis, there exist IAP no-verification vulnerabilities in these 28 apps. Moreover, Table 7 shows that there are 224 apps (including Mono-based and IL2CPP-based apps) adopting biometric data collection functions while having no permission requests in the manifest file. Specifically, 209 apps use the hand-tracking function while indicating no request for that permission in the `Androidmanifest.xml` file. Among them, 62 apps exploit face-tracking functions, 59 apps invoke eye-tracking functions, and 63 apps call body-tracking functions. This implies that *a significant number of apps do not adhere to the specifications of the Oculus VR app development documentation*. Biometric data collection can be enabled without user permission[2]. It exposes the risk of unknowingly stealing biometric data from users. It also indicates that Unity development framework for VR apps and the Oculus SDK have some potential vulnerabilities.

Answer to RQ3: Although only 5.60% of VR apps have used Unity IAP functions, all of them have IAP no-verification vulnerabilities. 49.12% of Unity VR apps have inconsistency between permission requests and biometric function usage, thereby causing leakage risks of human biometrics.

4.4 RQ4: To what extent is PII data leaked?

We also adopt a taint analysis of PII data leaks. Figure 7 reports the results of the taint analysis for PII data leakage by calculating a percentage of the number of source-to-sink paths found in each VR app. It can be found that calling from Activity-related methods such as `Activity`, `NativeActivity`, and `GameActivity` are the most popular source for obtaining PII data. The largest amount of PII data flows

[2]This issue was raised in the Meta community forum, but no explicit answer has been given until July 2023: https://communityforums.atmeta.com/t5/Oculus-Quest-2-and-Quest/Unity-Oculus-Integration-bug-Hand-tracking-always-enabled-no/td-p/753132

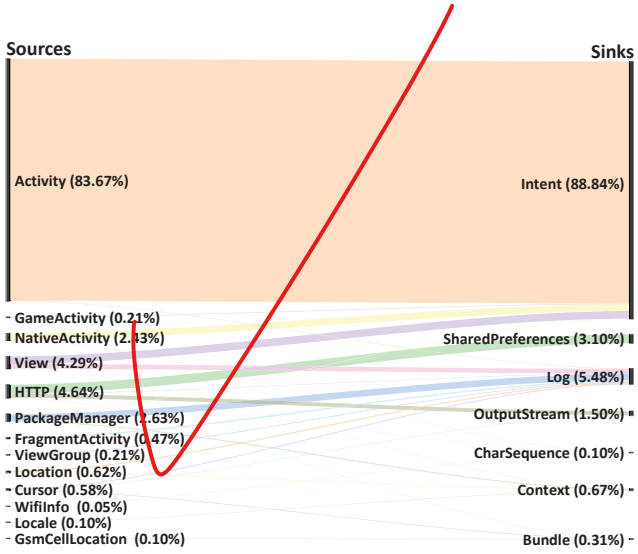


Figure 7: Data Leaks Detection by Taint Analysis

to the Intent-related and Log-related sinks, such as `content.Intent` and `util.Log`. The use of the sink method may cause data leaks. For example, in a virtual social app, there is a data flow from `Location` method to `registerReceiver()` in `content.Intent`. The BroadcastReceiver registered with the `registerReceiver()` method is global and exportable by default. If access is not restricted, it can be accessed by any external app, passing Intent to it to perform specific functions. Therefore, dynamically registered BroadcastReceiver may lead to security risks such as denial of service attacks, APP data leakage, or unauthorized calls [63].

Answer to RQ4: A number of VR apps have the leakage risks of PII sensitive data. Most data flow from activity-related methods to intent-related methods.

4.5 RQ5: How do the VR app developers comply with the privacy policies?

As mentioned in § 3.4, we collect policy statements from apps' privacy policies based on predefined ontologies by PolicyLint [3]. We check whether there are contradictory statements and GPDR violations in privacy policies and whether they are consistent with the app analysis results.

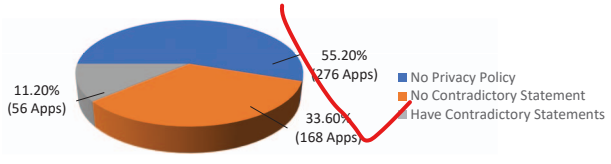


Figure 8: App Privacy Policy Distribution

As shown in Figure 8, unexpectedly 276 apps (55.20%) have no privacy policy though only 224 apps (44.80%) have privacy policies. Among 224 apps with privacy policies, 56 of them contain contradictory statements. For example, in a business and finance VR app, the privacy policy states that they do not sell personal information

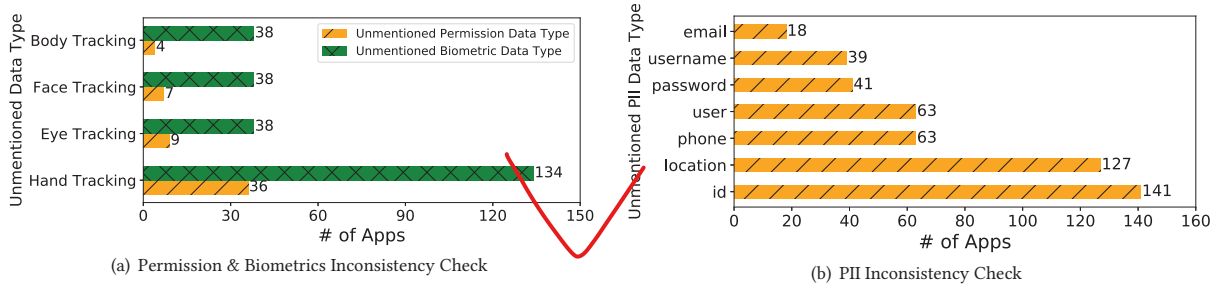
Table 8: GDPR Compliance Check

GDPR Violation Term	Risk Level	# Apps
Missing Social Media Clause	High	65
Missing Data Sharing Information	High	104
Missing GDPR Roles	High	133
Missing Data Subject Rights	High	60
Missing Legal Ground	High	134
Missing Data Retention Information	High	107
Missing Timestamp	High	45
Missing National Authority	High	109
Missing Sections	High	68
Missing GDPR Specificity	High	69
Missing Dedicated Privacy Mailbox	Medium	46
Missing Data Security Information	Medium	58
Missing Top Level Link	Medium	85
No Violation	N/A	38

to a third party while indicating that a third party may collect some specified category of personal information. It implies that there is still a certain percentage of VR apps containing unregulated privacy policies. Different from traditional mobile apps (Android apps on mobile phones), VR apps have higher chances to access highly-sensitive personal biometrics. Therefore, it is crucial to establish a consistent privacy policy for regulating VR app development.

Table 8 reports the GDPR compliance check results. We find that only 38 apps among these 224 apps with privacy policies have privacy policies fully complying with GDPR. Meanwhile, there are 13 GDPR violation terms in our check results: 10 high-risk terms and 3 medium-risk terms. Most app privacy policies (i.e., 134) miss the legal ground. The results imply that the normality of privacy policies of VR apps still needs to be improved. Privacy policies with no compliance with the law may expose developers to legal risks.

In contrast to conventional mobile apps, VR apps need access to massive PII data, including not only device id, name, and phone number, but also additional highly-sensitive human biometrics, such as hand coordinates, eye rotation, body shape, and face expressions (§ 3.4). We further check whether accessing to this PII sensitive data is explicitly mentioned in the corresponding privacy policy. In particular, according to the result of the permission inconsistency check from the manifest analysis shown in Figure 9(a), we find that 41 apps with privacy policies do not mention the usage of hand, eye, body, and face data while they are found to request these permissions by the manifest analysis. Meanwhile, 36 of these apps use hand-tracking data but do not state so in their privacy policies. According to the result of the PII inconsistency check from the decompiled Java and Smali codes, shown in Figure 9(b), we find that most apps with privacy policies (183 apps) do not mention the use of PII data in detail though they are found to use PII collection method in the code analysis. Moreover, 141 of these apps used *id* (e.g., device id) but do not mention it in their privacy policies. Further, we find that 136 apps have this inconsistency between biometric function usage and the privacy policy as shown in Figure 9(a). In addition, 134 of these apps identified hand-tracking data collection methods from the decompiled C# codes but do not mention them in their privacy policies. The results show that there



(a) Permission & Biometrics Inconsistency Check

(b) PII Inconsistency Check

Figure 9: Privacy Policy Inconsistency Check

exist a number of irregularities in the privacy policies, which have not been updated in time to address the privacy concerns of VR app data collection.

Answer to RQ5: Less than half of VR apps offer privacy policies though 25.00% of them contain contradictory statements. Meanwhile, 16.96% of 224 VR apps with privacy policies comply with GDPR regulations, 81.70% of them have no explicit mention of PII data usage in detail, and about 60% have inconsistency between human biometrics collection and privacy policy.

4.6 Discussion

We present two app cases to elaborate on security and privacy issues and offer several pieces of advice in VR app development.

Case Study. Considering the ethics issue, we have anonymized the specific names of the apps. *Case 1* (MD5 prefix: `4dbd4~`) is a virtual social VR app with over 5,200 downloads on SideQuest . We find that this App has no root detection implementation. The lack of root detection may lead to SDI vulnerability and data breaches. We also find that this app includes the usage of SQL raw query function. Due to the lack of root detection, the attacker can directly use administrator privileges to access/modify the database, resulting in user data leakage. Moreover, it also adopts insecure random generators, which may cause a predictable random number. Attackers can use predictable values to bypass permission verification. This app also allows clear text traffic, incurring a risk that a cyber attacker could eavesdrop on the transmitted data. In addition, despite the identified usage biometric data collection API, we do not find the corresponding `<uses-permission>` tag in the `Androidmanifest.xml` file, implying that the app does not apply for the permission while using the corresponding APIs. We also do not find a statement in our privacy policy with respect to the collection of such biometric data. The `id`, `phone`, and `location` data collection are not mentioned in the privacy policy while are adopted in the source code. In addition, the privacy policy contains GDPR violations, such as missing data sharing information and missing top-level link.

Case 2 (MD5 prefix: `c2300~`) is another virtual social VR app with more than 9,500 downloads . We find that the `allow_backup` flag in the manifest file is marked as *True*, which leads to a data leak risk. We also find that there exist some insecure encrypt functions by matching pattern `AES/ECB`, which indicates the app uses insecure ECB mode in the Cryptographic encryption algorithm. Besides, this app also utilizes some insecure hash functions including MD5-related hash functions (e.g., `Encrypter.MD5`), SHA-1-related functions (e.g., `Util.sha1`) and so on. A tracker called *Unity3d Ads* is also identified in this app. Besides, it lacks root detection. From the taint analysis result, we find that there exists PII data flow from `PackageManager` function to `Log` function. The use of `Log` function may expose network packet data to attackers and thus be intercepted by attackers for illegal activities in the metaverse such as harassment. The `location`, `password`, `user`, `username`, `phone`, and `location` data collection are not mentioned in the privacy policy but they are adopted in the source code.

Development Advice. Considering the findings in the case study, we offer some advice on the development of VR apps.

Advice 1: Set proper secure flags in the manifest file. Before releasing the app, it is suggested to set the security-related flags in the `Androidmanifest.xml` file as *False*, such as `allow_backup`, `debuggable`, and `use_cleartext_traffic`. If these flags are *True*, users' private behaviors can be inspected by attackers.

Advice 2: Enable root detection when starting the VR app. A rooted device may cause the association of app data and user data [51]. Consequently, invaded malware lurking in the VR device can steal users' private information. This can be even more dangerous in the metaverse, which involves users' frequent interactions (e.g., transactions of virtual assets).

Advice 3: Do not use insecure hash functions/encryption algorithms. Compared with traditional mobile apps, VR apps collect more diverse data (e.g., video and voice). It is crucial to use effective encryption methods, such as secure hash functions (e.g., SHA-256 [6]) and encryption methods, such as RSA with OAEP padding.

Advice 4: Check data flows used by the trackers. It is difficult to guarantee the security of these user data sent to third-party platforms by trackers. Therefore, developers should check the data flows used by the trackers to ensure no abuse or misuse.

Advice 5: Comply with permission requests to collect biometric data. Developers should comply with specifications for sensitive data collection, while app stores should strengthen code audits to prevent similar malware releases. Meanwhile, the sensitive data collection functions should be regulated to ensure the user's right to know how sensitive data is collected and prevent it from being misused.

Advice 6: Adapt privacy policies to fulfill VR apps' new features. Developers need to develop new privacy policies according to VR apps'

new features, such as privacy concerns with immersive social interactions and the collection of human biometrics. Further, the publication of a privacy policy is subject to relevant legislation.

5 LIMITATION AND THREATS TO VALIDITY

Limitation. The limitation of our research lies in the integrity of decompiled code. Since some apps are shelled for anti-cheating purposes, we cannot get the complete decompiled code to analyze the API call relationships in all apps. In addition, the data transmissions between different data-collection functions and the tracking algorithms is not open-sourced in the Oculus developer documentation. We will further analyze the call chains of these API functions in the future, especially for those of biometric data collection.

Threats to External Validity. The threats to external validity limit the scalability of our approach. In our biometric data propagation analysis, we mainly focus on VR apps developed based on Unity though there are other frameworks, such as UE [42], libGDX [49], and so on. Moreover, we only analyze VR apps working on Meta Oculus Quest 2 while there are some other popular VR/AR devices, such as the HTC VIVE Pro 2, Sony Playstation VR 2, and Pico 4, many of which are also Android or its variants (our tool may also apply to). In short, more types of VR apps need to undergo security and privacy assessments in the future.

Threats to Internal Validity. Although we collect 500 VR apps, which are almost 3 × of the state-of-the-art tool OVRSeen [53], we will evaluate more VR apps with the proliferation of VR and metaverse. Moreover, as for the static analysis for OS-related security and privacy vulnerabilities, we referred to pre-defined patterns based on [50] and the precision result they claimed is 96.19%. We manually checked the result of 20% of the apps and found no false positives, thus reducing the impact of tool accuracy. The taint analysis adopted in this work can also lead to some false positives, thereby affecting the accuracy of the results. To address this issue, we manually checked 20% of the identified paths and found no false positives, thus mitigating its side effect. With respect to another internal validity threat caused by the accuracy of the GDPRWise, we manually sampled 50 privacy policy cases to verify the accuracy of their GDPR compliance detection and found 9 false positives. We hypothesize three possible reasons: (i) the privacy policy is written in a non-English language (e.g., Japanese), which affects its detection accuracy; (ii) the link to the privacy policy is in PDF format, which is not supported by the service; and (iii) the link to the privacy policy contains additional external links. The extent of the impact needs further investigation.

6 RELATED WORK

Program Analysis of Mobile Apps. Many recent studies adopt both static tools and dynamic techniques to analyze the security and privacy vulnerabilities of mobile apps. Lee et al. [31] proposed a static tool to analyze inter-communication between Android Java and JavaScript codes. Sandeep [21] combined deep learning and static analysis to detect Android malware with high accuracy. Regarding dynamic analysis, Reardon et al. [43] constructed a testing environment to detect whether the app bypassed the permission model to access protected data. Huang et al. [22] proposed a testing framework based on net packet fuzzing for Android apps. In this paper, we use static analysis combined with privacy-policy analysis to perform security and privacy analysis on emerging VR apps.

Security and Privacy Analysis of VR Apps. With the rapid development of VR devices and metaverse platforms, the analysis of VR apps has received increasing attention. For example, Trimananda et al. [53] proposed a method, namely OVRSeen to analyze the privacy policies in Oculus VR apps by collecting network traffic and comparing them with the privacy policies although its dynamic analysis also has limited coverage for execution paths and unsoundness as indicated in [18]. Yarramreddy et al. [60] proposed a forensic analysis of VR social apps to reveal some forensically relevant data from network traffic and the VR systems. Casey et al. [10] discovered a new attack against VR systems, which can open the VR camera without user permission and insert images into users' vision to distract users' attention in a virtual environment. This paper focuses on metaverse-related VR apps with a comprehensive assessment of their security and privacy status.

Unity-based VR Apps. Since most VR apps have been developed based on Unity to render 3D environment, achieve immersive user experience, and collect biometric data, we also review related work as follows. Shim et al. [47] proposed a reverse engineering method with a combination of static and dynamic analysis to analyze malicious Unity apps. This tool can be used to analyze the native code of Java, C, C++, and the Mono layer where the C# code runs. Volokh et al. [55] proposed a Unity game code logic analysis tool based on static analysis, which can be used to provide an available action state set at a game state for players. Zuo et al. [64] conducted an in-depth analysis of the security of paid implementations in Unity-based handheld games by designing and implementing the static tool, namely PaymentScope to automatically identify vulnerable IAPs in mobile games. In this paper, we design a variant of PaymentScope to detect not only vulnerable IAPs but also biometric data usage.

7 CONCLUSION

With the proliferation of diverse VR devices and the increasing attention of the metaverse in recent years, VR apps have received a boosted development and proliferation. Although numerous VR apps have been released, little attention has been paid to the security and privacy issues of emerging VR apps. In this paper, we have developed a security and privacy assessment tool, namely the VR-SP detector for VR apps. The VR-SP detector has been implemented with the integration of program static analysis and privacy policy analysis methods. Using the VR-SP detector, we have conducted the security and privacy assessment of 500 popular VR apps. Our analytical results have revealed important security and privacy issues of existing metaverse-related VR apps. Based on our findings, we have made development recommendations for future VR apps with security and privacy preservation.

ACKNOWLEDGMENTS

The work described in this paper is partially supported by the National Natural Science Foundation of China (62032025), COMP Department Start-up Fund of Hong Kong Baptist University (HKBU), Faculty Start-up Grant for New Academics of HKBU, SD/COMP Joint Research Scheme (ID: P0042739), and Departmental Incentive Scheme of HKBU COMP. We would like anonymous reviewers for their constructive comments.

REFERENCES

[1] 2020. *dnSpy*. https://github.com/dnSpy/dnSpy

[2] Maha Alghawazi, Daniyal Alghazzawi, and Suaad Alarifi. 2022. Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review. *Journal of Cybersecurity and Privacy* 2, 4 (2022), 764–777. https://doi.org/10.3390/jcp2040039

[3] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. 2019. PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 585–602. https://www.usenix.org/conference/usenixsecurity19/presentation/andow

[4] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. 2020. Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 985–1002. https://www.usenix.org/conference/usenixsecurity20/presentation/andow

[5] Vincent Ang and Lwin Khin Shar. 2021. COVID-19 One Year on – Security and Privacy Review of Contact Tracing Mobile Apps. *IEEE Pervasive Computing* 20, 4 (2021), 61–70. https://doi.org/10.1109/MPRV.2021.3115478

[6] Andrew W. Appel. 2015. Verification of a Cryptographic Primitive: SHA-256. *ACM Trans. Program. Lang. Syst.* 37, 2, Article 7 (apr 2015), 31 pages. https://doi.org/10.1145/2701415

[7] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14)*. Association for Computing Machinery, New York, NY, USA, 259–269. https://doi.org/10.1145/2594291.2594299

[8] Jaime Benjumea, Jorge Ropero, Octavio Rivera-Romero, Enrique Dorronzoro-Zubiete, and Alejandro Carrasco. 2020. Assessment of the Fairness of Privacy Policies of Mobile Health Apps: Scale Development and Evaluation in Cancer Apps. *JMIR Mhealth Uhealth* 8, 7 (28 Jul 2020), e17134. https://doi.org/10.2196/17134

[9] GDPRWise BV. 2023. *GDPRWise Policy Checker*. https://gdprwise.eu/policy-checker/

[10] Peter Casey, Ibrahim Baggili, and Ananya Yarramreddy. 2021. Immersive Virtual Reality Attacks and the Human Joystick. *IEEE Transactions on Dependable and Secure Computing* 18, 2 (2021), 550–562. https://doi.org/10.1109/TDSC.2019.2907942

[11] Cheng Chang, Huaxin Li, Yichi Zhang, Suguo Du, Hui Cao, and Haojin Zhu. 2019. Automated and Personalized Privacy Policy Extraction Under GDPR Consideration. In *Wireless Algorithms, Systems, and Applications*, Edoardo S. Biagioni, Yao Zheng, and Siyao Cheng (Eds.). Springer International Publishing, Cham, 43–54.

[12] Ruizhi Cheng, Nan Wu, Songqing Chen, and Bo Han. 2022. Reality Check of Metaverse: A First Look at Commercial Social Virtual Reality Platforms. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 141–148. https://doi.org/10.1109/VRW55335.2022.00040

[13] Alvin Christopher Santoso and Petrus Santoso. 2022. Aplikasi Ruangan Maya Berbasis Android OS pada Headset Virtual Reality Oculus Quest 2. *Jurnal FORTECH* 3, 2 (Sep. 2022), 51–56. https://doi.org/10.56795/fortech.v3i2.321

[14] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. 2016. A Survey of Man In The Middle Attacks. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 2027–2051. https://doi.org/10.1109/COMST.2016.2548426

[15] Carlos Cortés, Pablo Pérez, and Narciso García. 2019. Unity3D-based app for 360VR subjective quality assessment with customizable questionnaires. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*. 281–282. https://doi.org/10.1109/ICCE-Berlin47944.2019.8966170

[16] Hesham Darvish and Mohammad Husain. 2018. Security Analysis of Mobile Money Applications on Android. In *2018 IEEE International Conference on Big Data (Big Data)*. 3072–3078. https://doi.org/10.1109/BigData.2018.8622115

[17] Anthony Desnos and G Gueguen. 2018. *Androguard documentation*. https://androguard.readthedocs.io/en/latest/

[18] David Devecsery, Peter M. Chen, Jason Flinn, and Satish Narayanasamy. 2018. Optimistic Hybrid Analysis: Accelerating Dynamic Analysis through Predicated Static Analysis. *SIGPLAN Not.* 53, 2 (mar 2018), 348–362. https://doi.org/10.1145/3296957.3177153

[19] Ibrahim F. Elashry, Osama S. Farag Allah, Alaa M. Abbas, and S. El-Rabaie. 2009. A new diffusion mechanism for data encryption in the ECB mode. In *2009 International Conference on Computer Engineering & Systems*. 288–293. https://doi.org/10.1109/ICCES.2009.5383254

[20] Inc. Epic Games. [n. d.]. *Unreal Engine*. https://www.unrealengine.com/ (2023, July 24).

[21] Sandeep HR. 2019. Static Analysis of Android Malware Detection using Deep Learning. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. 841–845. https://doi.org/10.1109/ICCS45141.2019.9065765

[22] Xinyue Huang, Anmin Zhou, Peng Jia, Luping Liu, and Liang Liu. 2019. Fuzzing the Android Applications With HTTP/HTTPS Network Data. *IEEE Access* 7 (2019), 59951–59962. https://doi.org/10.1109/ACCESS.2019.2915339

[23] Yan Huang, Yi Joy Li, and Zhipeng Cai. 2023. Security and Privacy in Metaverse: A Comprehensive Survey. *Big Data Mining and Analytics* 6, 2 (2023), 234–247. https://doi.org/10.26599/BDMA.2022.9020047

[24] James P. Hughes and Whitfield Diffie. 2022. The Challenges of IoT, TLS, and Random Number Generators in the Real World: Bad Random Numbers Are Still with Us and Are Proliferating in Modern Systems. *Queue* 20, 3 (jul 2022), 18–40. https://doi.org/10.1145/3546933

[25] Jing Hui, Yueliang Zhou, Mohamed Oubibi, Weifeng Di, Lixin Zhang, and Sijia Zhang. 2022. Research on Art Teaching Practice Supported by Virtual Reality (VR) Technology in the Primary Schools. *Sustainability* 14, 3 (2022). https://doi.org/10.3390/su14031246

[26] Thien Huynh-The, Quoc-Viet Pham, Xuan-Qui Pham, Thanh Thi Nguyen, Zhu Han, and Dong-Seong Kim. 2023. Artificial intelligence for the metaverse: A survey. *Engineering Applications of Artificial Intelligence* 117 (2023), 105581. https://doi.org/10.1016/j.engappai.2022.105581

[27] Fortune Business Insights. 2023. *Virtual Reality Market Size, Share and COVID-19 Impact Analysis, By Component (Hardware, Software, and Content), By Device Type (Head Mounted Display (HMD), VR Simulator, VR Glasses, Treadmills and Haptic Gloves, and Others), By Industry (Gaming, Entertainment, Automotive, Retail, Healthcare, Education, Aerospace and Defense, Manufacturing, and Others), and Regional Forecast, 2023-2030*. https://www.fortunebusinessinsights.com/industry-reports/virtual-reality-market-101378

[28] Konrad Kollnig, Pierre Dewitte, Max Van Kleek, Ge Wang, Daniel Omeiza, Helena Webb, and Nigel Shadbolt. 2021. A Fait Accompli? An Empirical Study into the Absence of Consent to Third-Party Tracking in Android Apps. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*. USENIX Association, 181–196. https://www.usenix.org/conference/soups2021/presentation/kollnig

[29] Grace LaMalva and Suzanna Schmeelk. 2020. MobSF: Mobile Health Care Android Applications Through The Lens of Open Source Static Analysis. In *2020 IEEE MIT Undergraduate Research Technology Conference (URTC)*. 1–4. https://doi.org/10.1109/URTC51696.2020.9668870

[30] Jungmi Lee. 2022. A study on the intention and experience of using the metaverse. *Jahr: Europski časopis za bioetiku* 13, 1 (2022), 177–192.

[31] Sungho Lee, Julian Dolby, and Sukyoung Ryu. 2016. HybriDroid: Static Analysis Framework for Android Hybrid Applications. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE '16)*. Association for Computing Machinery, New York, NY, USA, 250–261. https://doi.org/10.1145/2970276.2970368

[32] Douglas J. Leith and Stephen Farrell. 2021. Contact Tracing App Privacy: What Data Is Shared By Europe's GAEN Contact Tracing Apps. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488728

[33] He Li, Lu Yu, and Wu He. 2019. The Impact of GDPR on Global Technology Development. *Journal of Global Information Technology Management* 22, 1 (2019), 1–6. https://doi.org/10.1080/1097198X.2019.1569186 arXiv:https://doi.org/10.1080/1097198X.2019.1569186

[34] Song Liao, Christin Wilson, Long Cheng, Hongxin Hu, and Huixing Deng. 2020. Measuring the Effectiveness of Privacy Policies for Voice Assistant Applications. In *Annual Computer Security Applications Conference* (Austin, USA) *(ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 856–869. https://doi.org/10.1145/3427228.3427250

[35] Zhuo Ma, Haoran Ge, Yang Liu, Meng Zhao, and Jianfeng Ma. 2019. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. *IEEE Access* 7 (2019), 21235–21245. https://doi.org/10.1109/ACCESS.2019.2896003

[36] Tahrima Mustafa, Richard Matovu, Abdul Serwadda, and Nicholas Muirhead. 2018. Unsure How to Authenticate on Your VR Headset? Come on, Use Your Head!. In *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics* (Tempe, AZ, USA) *(IWSPA '18)*. Association for Computing Machinery, New York, NY, USA, 23–30. https://doi.org/10.1145/3180445.3180450

[37] Stylianos Mystakidis. 2022. Metaverse. *Encyclopedia* 2, 1 (2022), 486–497. https://doi.org/10.3390/encyclopedia2010031

[38] Huansheng Ning, Hang Wang, Yujia Lin, Wenxi Wang, Sahraoui Dhelim, Fadi Farha, Jianguo Ding, and Mahmoud Daneshmand. 2021. A Survey on Metaverse: the State-of-the-art, Technologies, Applications, and Challenges. *arXiv preprint arXiv:2111.09673* (2021).

[39] Fariha Nusrat, Foyzul Hassan, Hao Zhong, and Xiaoyin Wang. 2021. How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 473–485. https://doi.org/10.1109/ICSE43902.2021.00052

[40] Marten Oltrogge, Nicolas Huaman, Sabrina Amft, Yasemin Acar, Michael Backes, and Sascha Fahl. 2021. Why Eve and Mallory Still Love Android: Revisiting TLS (In)Security in Android Applications. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 4347–4364. https://www.usenix.org/

conference/usenixsecurity21/presentation/oltrogge
[41] Sang-Min Park and Young-Gab Kim. 2022. A Metaverse: Taxonomy, Components, Applications, and Open Challenges. *IEEE Access* 10 (2022), 4209–4251. https://doi.org/10.1109/ACCESS.2021.3140175
[42] Weichao Qiu and Alan Yuille. 2016. UnrealCV: Connecting Computer Vision to Unreal Engine. In *Computer Vision – ECCV 2016 Workshops*, Gang Hua and Hervé Jégou (Eds.). Springer International Publishing, Cham, 909–916.
[43] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 603–620. https://www.usenix.org/conference/usenixsecurity19/presentation/reardon
[44] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. 2015. Towards Discovering and Understanding Task Hijacking in Android. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Washington, D.C.) *(SEC'15)*. USENIX Association, USA, 945–959.
[45] Ashish Rajendra Sai, Jim Buckley, and Andrew Le Gear. 2019. Privacy and Security Analysis of Cryptocurrency Mobile Applications. In *2019 Fifth Conference on Mobile and Secure Services (MobiSecServ)*. 1–6. https://doi.org/10.1109/MOBISECSERV.2019.8686583
[46] Faysal Hossain Shezan, Syeda Farzia Afroze, and Anindya Iqbal. 2017. Vulnerability detection in recent Android apps: An empirical study. In *2017 International Conference on Networking, Systems and Security (NSysS)*. 55–63. https://doi.org/10.1109/NSysS.2017.7885802
[47] Jaewoo Shim, Kyeonghwan Lim, Seong-je Cho, Sangchul Han, and Minkyu Park. 2018. Static and dynamic analysis of Android malware and goodware written with unity framework. *Security and Communication Networks* 2018 (2018).
[48] Shao Shuai, Dong Guowei, Guo Tao, Yang Tianchang, and Shi Chenjie. 2014. Modelling Analysis and Auto-detection of Cryptographic Misuse in Android Applications. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. 75–80. https://doi.org/10.1109/DASC.2014.22
[49] Lee Stemkoski. 2015. *The LibGDX Framework*. Apress, Berkeley, CA, 13–46. https://doi.org/10.1007/978-1-4842-1500-5_2
[50] Ruoxi Sun, Wei Wang, Minhui Xue, Gareth Tyson, Seyit Camtepe, and Damith C. Ranasinghe. 2021. An Empirical Assessment of Global COVID-19 Contact Tracing Applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1085–1097. https://doi.org/10.1109/ICSE43902.2021.00101
[51] San-Tsai Sun, Andrea Cuadros, and Konstantin Beznosov. 2015. Android Rooting: Methods, Detection, and Evasion. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (Denver, Colorado, USA) *(SPSM '15)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/2808117.2808126
[52] Unity Technologies. [n. d.]. *Unity documentation - 2d or 3d projects*. https://docs.unity3d.com/ (2023, March 24).
[53] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. 2022. OVRseen: Auditing Network Traffic and Privacy Policies in Oculus VR. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3789–3806. https://www.usenix.org/conference/usenixsecurity22/presentation/trimananda
[54] Luca Verderame, Davide Caputo, Andrea Romdhana, and Alessio Merlo. 2020. On the (Un)Reliability of Privacy Policies in Android Apps. In *2020 International Joint Conference on Neural Networks (IJCNN)*. 1–9. https://doi.org/10.1109/IJCNN48605.2020.9206660
[55] Sasha Volokh and William G.J. Halfond. 2022. Static Analysis for Automated Identification of Valid Game Actions During Exploration. In *Proceedings of the 17th International Conference on the Foundations of Digital Games* (Athens, Greece) *(FDG '22)*. Association for Computing Machinery, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/3555858.3555898
[56] Martin Vondráček, Ibrahim Baggili, Peter Casey, and Mehdi Mekni. 2023. Rise of the Metaverse's Immersive Virtual Reality Malware and the Man-in-the-Room Attack & Defenses. *Computers & Security* 127 (2023), 102923. https://doi.org/10.1016/j.cose.2022.102923
[57] Isabell Wohlgenannt, Alexander Simons, and Stefan Stieglitz. 2020. Virtual reality. *Business & Information Systems Engineering* 62 (2020), 455–461.
[58] Yi Wu, Cong Shi, Tianfang Zhang, Payton Walker, Jian Liu, Nitesh Saxena, and Yingying Chen. 2023. Privacy Leakage via Unrestricted Motion-Position Sensors in the Age of Virtual Reality: A Study of Snooping Typed Input on Virtual Keyboards. In *2023 IEEE Symposium on Security and Privacy (SP)*. 3382–3398. https://doi.org/10.1109/SP46215.2023.10179301
[59] Xiaoyi Yang and Xueling Zhang. 2023. A Study of User Privacy in Android Mobile AR Apps. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 226, 5 pages. https://doi.org/10.1145/3551349.3560512
[60] Ananya Yarramreddy, Peter Gromkowski, and Ibrahim Baggili. 2018. Forensic Analysis of Immersive Virtual Reality Social Applications: A Primary Account. In *2018 IEEE Security and Privacy Workshops (SPW)*. 186–196. https://doi.org/10.1109/SPW.2018.00034
[61] Sophia Yoo and Xiaoqi Chen. 2021. Secure Keyed Hashing on Programmable Switches. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Secure Programmable Network INfrastructure* (Virtual Event, USA) *(SPIN '21)*. Association for Computing Machinery, New York, NY, USA, 16–22. https://doi.org/10.1145/3472873.3472881
[62] Hang Zhang, Dongdong She, and Zhiyun Qian. 2015. Android Root and Its Providers: A Double-Edged Sword. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1093–1104. https://doi.org/10.1145/2810103.2813714
[63] Hao Zhou, Haoyu Wang, Yajin Zhou, Xiapu Luo, Yutian Tang, Lei Xue, and Ting Wang. 2021. Demystifying Diehard Android Apps. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 187–198. https://doi.org/10.1145/3324884.3416637
[64] Chaoshun Zuo and Zhiqiang Lin. 2022. Playing Without Paying: Detecting Vulnerable Payment Verification in Native Binaries of Unity Mobile Games. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3093–3110. https://www.usenix.org/conference/usenixsecurity22/presentation/zuo