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

Hanyang Guo, Hong-Ning Dai, *Senior Member, IEEE*, Xiapu Luo, *Senior Member, IEEE*, Gengyang Xu, Fengliang He, and Zibin Zheng, *Fellow, IEEE*

***Abstract*—Virtual Reality (VR) has accelerated its prevalent adoption in emerging metaverse applications, but it is not a fundamentally new technology. On the one hand, most VR operating systems (OS) are based on off-the-shelf mobile OS (e.g., Android OS). As a result, VR apps also inevitably inherit privacy and security deficiencies from conventional mobile apps. On the other hand, in contrast to traditional mobile apps, VR apps can achieve an immersive experience via diverse VR devices, such as head-mounted displays, body sensors, and controllers. However, 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 900 popular VR apps. We obtain the original apps from the popular SideQuest app store and extract Android PacKage (APK) files via the Meta Quest 2 device. We evaluate the security vulnerabilities and privacy data leaks of these VR apps through VR app analysis, taint analysis, privacy policy analysis, and user review 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. Further, user reviews also indicate their privacy concerns about relevant biometric data. Based on these findings, we make suggestions for the future development of VR apps.**

***Index Terms*—Virtual reality, metaverse, static analysis, security and privacy.**

I. Introduction

VIRTUAL Reality (VR) [1] 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 [2], 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 [3] 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 generate 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 Quest 2 [4] (former name: Oculus Quest 2) and ByteDance's Pico 4 [5]). 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. In addition, VR apps also adopt third-party libraries (TPLs) like conventional mobile apps to implement specific app functionalities or add new features.

Despite the similarity to Android mobile apps, VR apps have unique characteristics different from conventional mobile apps.

Received 3 July 2024; revised 11 January 2025; accepted 12 March 2025. Date of publication 19 March 2025; date of current version 16 May 2025. This work was supported in part by Hong Kong Baptist University Seed Funding for Collaborative Research under Grant RC-SFCRG/23-24/R2/SCI/06, in part by HKPolyU Grant (H-ZGGG), and in part by CCF-Sangfor "Yuanwang" Research Fund. Recommended for acceptance by J. Garcia. *(Corresponding author: Hong-Ning Dai.)*

Hanyang Guo is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong 999077, China, and also with the School of Software Engineering, Sun Yat-sen University, Zhuhai 519082, China (e-mail: guohy36@mail2.sysu.edu.cn).

Hong-Ning Dai, Gengyang Xu, and Fengliang He are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong 999077, China (e-mail: hndai@ieee.org; 21253277@life.hkbu.edu.hk; csflhe@comp.hkbu.edu.hk).

Xiapu Luo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong 999077, China (e-mail: csxluo@comp.polyu.edu.hk).

Zibin Zheng is with the School of Software Engineering, Sun Yat-sen University, Zhuhai 519082, China (e-mail: zhzibin@mail.sysu.edu.cn).

Digital Object Identifier 10.1109/TSE.2025.3553283

(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., Meta 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. Using 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. (v) User reviews of VR apps may have additional privacy concerns about biometrics information, such as access to information permissions.

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 [6]. Ignorance of these emerging security risks will greatly dampen users' enthusiasm for using VR apps and purchasing VR devices as well as VR services [7], [8]. 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 [9].

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, privacy-policy analysis, and review analysis. Despite a recent study (OVRSEEN) on analyzing privacy policies of VR apps [10], it mainly focuses on network traffic rather than the implemented codes of VR apps. Our VR-SP detector integrates both static code analysis tools, privacy-policy analysis techniques, and review analysis techniques, thereby effectively revealing security and privacy risks. Based on the VR-SP detector, we conduct a comprehensive empirical study on 900 popular VR apps (i.e., more than 5 $\times$ of OVRSEEN and 1.8 $\times$ of our preliminary study [11]) from the largest VR app store SideQuest.

After extracting the original APK file for each app after installing it in Meta 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, vulnerable TPLs, and dangerous permission usage in TPLs 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 [12], [13], we hence leverage a static binary analysis framework to detect the flows, biometric data usage, and IAP data in Unity-developed VR apps. Also, we adopt a privacy policy analysis based on natural language processing (NLP) to detect inconsistencies in the privacy policies of VR apps. At last, we collect the user reviews of the VR apps from the SideQuest app store and analyze the concerns about VR apps. We have obtained many insightful findings, on which we provide some suggestions for developing 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 (i.e., PII and biometric) data leaks.
- Our tool integrates static analysis and privacy-policy analysis technologies. Specifically, 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. Holistically, the proposed tool also adopts user review analysis.
- We run our tool on 900 popular VR apps and find that more than 95.40% of the apps have no root detection (No RD) vulnerability and 37.20% exist insecure random generator (IRG) vulnerability. Only one VR app uses the vulnerable TPL and 22.00% use general TPLs but there is also only one app that involves calling dangerous permissions in the TPL. Moreover, 44.40% of the apps invoke functions using biometric data though there are no requests in the manifest file. In addition, 55.20% of apps have no privacy policy and 11.00% of apps have contradictory statements in privacy policies. As for user reviews, about 6.15% of reviews concern about dangerous permission usage, and 3.61% of them are biometric permission-related.
- Based on the findings, we provide some advice on VR app development. We make our tool publicly available at https://github.com/Henrykwokkk/Meta-detector-expand.

II. Background of VR Apps

A. 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, we mainly consider the following five types of VR apps:

- **Virtual society** [14], [15]. These VR apps provide users with virtual spaces to interact and socialize with others. Typical apps include Rec Room, VRChat, etc.
- **Gaming** [15], [16]. A large body of VR apps are themed with games, such as Pavlov VR, Echo VR, etc.
- **Art and culture** [17]. Many VR apps support virtual museums, exhibitions, concerts, and other cultural and artistic events, such as Forbidden City Journey, Gravity Sketch, etc.
- **Education** [15]. 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** [18]. Many VR apps provide virtual stores, trading platforms, and financial services, e.g., Decentraland.

B. VR App Security and Privacy Vulnerabilities

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 [19], [20].

Insecure Flag Settings: These vulnerabilities come from insecure flags in the configuration file `AndroidManifest.xml`, such as `allowBackup`, `debuggable`, and `clearTextTraffic`. We next elaborate on them as follows.

(i) `allowBackup` [21]: The `allowBackup` flag is used to specify whether or not the app's data is allowed to be backed up and restored through Android's backup mechanism. Positively setting the `allowBackup` flag (i.e., setting to `true`) allows the user's data to be stored in the cloud, which can nevertheless lead to the potential disclosure of sensitive data, especially when the application stores the user's private information or sensitive data.

(ii) `debuggable` [22]: The `debuggable` flag is used to specify whether the application can be debugged. In a debug build, it is usually set to `true` and it is set to `false` in the release build. A positive setting in the release build means that an attacker can use debugging tools to peek into the internal running state of the application, variable values, etc, thereby revealing sensitive information or making it easier for an attacker to find vulnerabilities in the application.

(iii) `clearTextTraffic` [23]: The `clearTextTraffic` flag is used to specify whether the application allows plaintext (unencrypted) HTTP traffic. If it is set to `true`, apps can use plaintext HTTP traffic, thus making apps be vulnerable to man-in-the-middle attacks (MITM) [24] because unencrypted data can be intercepted and tampered with in transit.

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 permission requests from other peripheral VR-related devices, e.g., controllers and keyboards.

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 [19], [25], [26], [27] also presented general vulnerabilities. We summarize them into the following nine categories in the VR context.

(1) *SQL Database Injection (SDI) in VR apps* [28] means that an attacker can insert additional SQL statements to the end of a pre-defined 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* [29] 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)* [30] 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)* [31] 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* [33] 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) [34] and Insecure Hash Functions (IHF) [35]. (8) *Root Detection (RD) of VR devices* [36] 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* [37] include not only trackers in general Android apps (e.g., *Google Firebase Analytics*) but also other VR-specified trackers (e.g., *Unity3d Ads*).

Third-Party Libraries Usage: VR apps also adopt TPLs to implement different functionalities and improve development efficiency [38]. Similar to Android apps, the usage of TPLs may cause some security and privacy risks [39]. For example, it has been demonstrated that third-party libraries in Android applications pose risks to security and privacy by introducing vulnerabilities into the apps they integrate with or by improperly exploiting the permissions granted to them [40].

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 [41] and Unreal Engine (UE) [42] 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,

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.

- **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, Meta 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**: 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.
- **New Features of User Reviews**: Since VR apps request other types of data such as biometric data, users may have concerns about these new types of data and their access rights, such as concerns about their security and user experience.

III. Methodology of VR-SP Detector

This section elaborates on the proposed VR-SP detector to analyze Meta VR apps. Fig. 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) PII Data Leaks Identification, (4) Privacy Policy Analysis, and (5) Review Analysis. The integration of VR app analysis and PII data leaks identification provides a comprehensive security and privacy assessment. Privacy policy analysis is integrated with VR app analysis and PII data leaks identification to execute norm and consistency checks. Review analysis is adopted to ensure the impact of security and privacy assessment results. The details are described as follows.

A. App Collection

According to rankings and popularity, we collect the 900 most popular VR apps from both the official Meta VR app store and SideQuest. The latter is the most popular third-party store endorsed by Meta. Compared with the Meta store, SideQuest has received 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 Meta VR app store or SideQuest. We then install them on Meta 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., Meta Quest Pro). These apps are collected according to the categories specified in Section II-A. We obtained VR apps by the popularity and the keywords specified in Table I. We use the "hot" ranking in SideQuest as the popularity metric and consider only those free apps. At last, we collected 900 apps including 109 virtual social apps, 503 game apps, 128 art and culture apps, 120 education apps, and 40 business and finance apps (the distribution of those apps is given in Section IV).

After collecting apps, we then extract APK files from those apps. Since neither the Meta official store nor SideQuest store provides a direct downloading link for APK files, we extract APK files by installing each app on the Meta Quest 2 device. We first connect Meta Quest 2 to the PC via a USB cable and install the app into Meta 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 Section III-D) and review analysis (in Section III-E), we also collect the policy statement and user review of each of those 900 apps.

B. VR App Analysis

The goal of the VR app analysis module is to detect general security and privacy vulnerabilities in codes and configuration files. 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 [43], which is an open-source Python tool capable of extracting different kinds of information from the individual components of an APK file [44]. Supporting multiple versions of Android and DEX file formats, Androguard can parse and analyze most Android applications. It also supports plug-ins and scripting extensions that allow users to customize

[1]We mainly consider these functions: `OVRHand.OVRMesh.IOVRMesh- DataProvider.GetMeshType`, `OVRBody.OVRSkeletonRenderer.IOVR- SkeletonRendererDataProvider.GetSkeletonRendererData`, `VREye Gaze.CalculateEyeRotation`, and `VRFaceExpressions.ToArray`, etc.

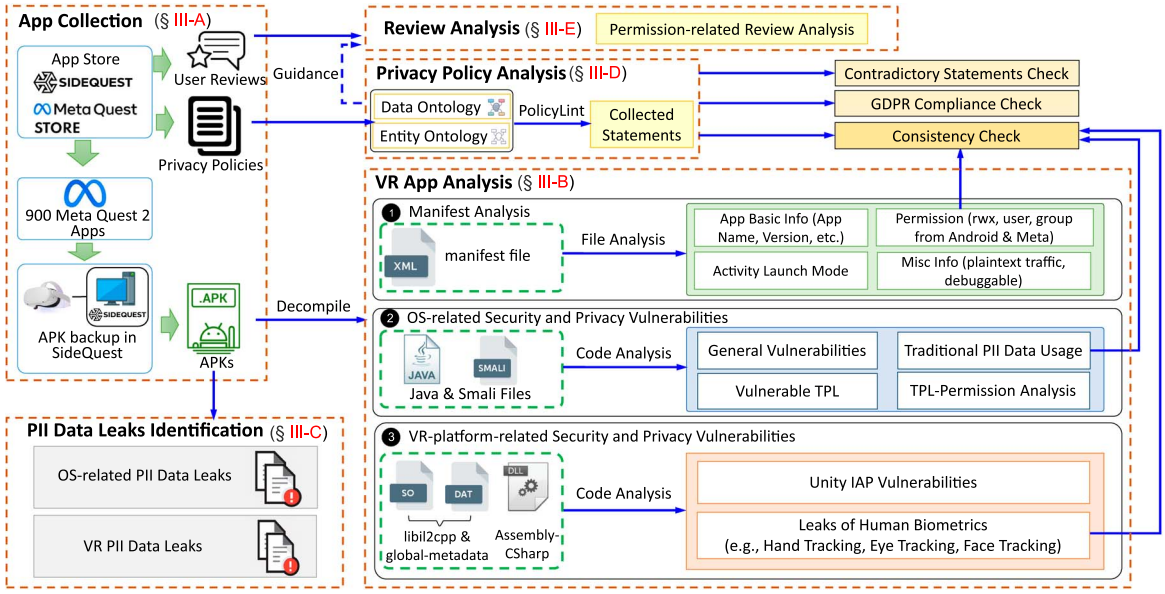


Fig. 1. The Overview of *VR-SP Detector.*

TABLE I
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

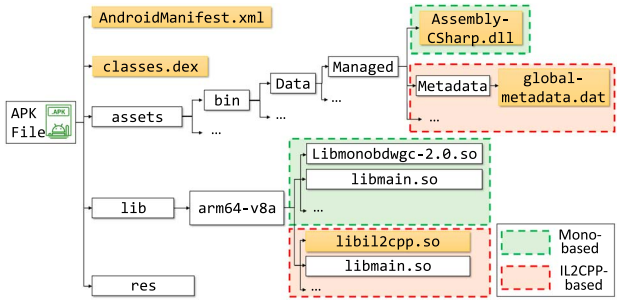


Fig. 2. APK structure of VR App.

and extend the tool's functionality as needed to improve analysis efficiency. Therefore, it has good robustness in analyzing VR-specific apps. With an APK file as input, Androguard can parse the structure of APK files to obtain configuration files (e.g., `AndroidManiest.xml` file), `classes.dex` file and so on. Fig. 2 shows an example of the APK structure of VR apps. It can also parse detailed information in each file and analyze the code logic and the data flow.

Obtaining the decompiled files, we then 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 Fig. 1. The detailed analysis procedure is elaborated as follows.

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., Meta 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 `uses_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 Meta (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 Meta 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) [45].

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` by adopting Androguard. The `classes.dex` file contains the bytecode, on which the Dalvik virtual machine runs. Androguard can parse the bytecode into a more manageable internal representation. This representation format typically includes parsing the headers, string pools, type pools, field pools, method pools, and class definitions in `classes.dex` files. It can also construct the method (function) call graph. Although the parsed result can not provide full and detailed variable name and method signature information since some apps adopt code obfuscation techniques [46], it can be used to detect some general security and privacy vulnerabilities indicated in Section II-B1 in VR apps. Specifically, we use pre-defined patterns based on [19] to detect potentially vulnerable methods (functions) and strings from Java codes or parsed Dalvik bytecodes. We then search whether the function call paths exist from the call graph. 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 [47]. In summary, there are nine types of security and privacy vulnerabilities that we detect in VR apps. Table II 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 Section III-D).

As for TPL detection, nowadays there exist different types of TPL detection tools such as LibScout [39], Libradar [38], LibD [48], and so on. However, most of these tools are outdated as they have no longer been maintained. Meanwhile, the version of Android OS they detected was not compatible with the latest version of Meta Horizon OS (based on Android 12). To address this challenge, we adopt a general-purpose TPL detection tool called LibScan [49]. This tool detects TPLs based on method-opcode similarity and call-chain-opcode similarity. We adopt this tool to detect 205 vulnerable TPLs collected by [49] in VR apps. Moreover, we also detect the usage of 255 different general TPLs collected by [49] and [40]. Thereafter, we attain some mappings between permission and APIs by searching permission keywords [50]. We search these APIs in the detected TPL packages so as to obtain improper permission grants.

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 Section II-B2, VR apps adopt game engines to achieve immersive features. We mainly consider Unity, which is the most dominating framework in VR software development [51]. We adopt a static native binary analysis tool [13] 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 Fig. 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 Fig. 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* [52] 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 [13], 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. Taint analysis is a static program analysis technique used to detect potential security vulnerabilities in programs. It identifies possible security problems by tracing the propagation path of data from untrustworthy sources (taint sources) to sensitive operations (taint sinks) [53]. Fig. 3 shows the framework of IAP vulnerability detection. The details are shown as follows.

- (i) Identification of tainted sources about payment & Marking tainted data of payment: Tainted sources are those places in a program, which receive sensitive

TABLE II
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.

data inputs. These inputs may come from external sources such as the user, the network, the file system, etc. When sensitive data enters the program from a tainted source, the data is marked as tainted data, implying that the data needs to be tracked along its propagation path. In our task, We define the method `UnityEngine.Purchasing.Product.get_receipt` as the tainted sources and the return value as tainted data of payment.

- (ii) Identification of tainted sinks about payment validation: Tainted sinks are those places in a program that perform sensitive operations. 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 tainted sinks.
- (iii) Taint propagation of payment data: Once payment data is marked as "tainted", the taint analysis tool traces the propagation path of that data through the program. This includes variable assignments, function calls, manipulation of data structures, etc. The propagation rules usually include: ❶ If tainted data is assigned to another variable, then that variable is also marked as tainted. ❷ If tainted data is passed to a function parameter, then that parameter is also marked as tainted. ❸ If tainted data is stored in a data structure (e.g., array, object), then the relevant part of that data structure is also marked as tainted. The construction of the payment data propagation path is executed based on data flow analysis.
- (iv) IAP vulnerabilities Detection: By exploiting the propagation path of tainted data, 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 Section II-B2 according to the Meta development document. 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 Section III-D).

C. 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 referring to [53]. The PII data leaks identification module further analyzes the propagation of PII data on the basis of the VR app analysis module presented in Section III-B, thus further complementing the analysis of privacy leakage risks for a more comprehensive security and privacy assessment. Fig. 3 shows the framework of PII data leaks identification. The details of its taint analysis are similar to those of IAP vulnerabilities shown as follows.

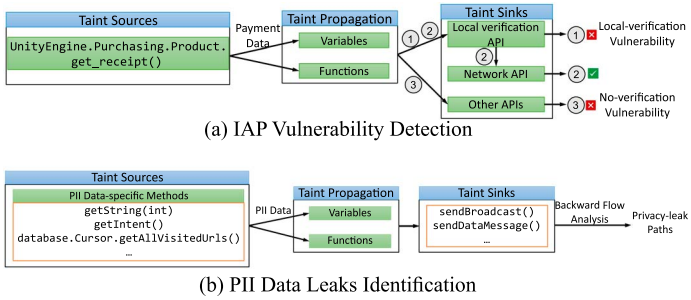


Fig. 3. Taint analysis frameworks.

- (i) Identification of VR-related PII data taint sources: We firstly 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*). These data are regarded as sensitive data inputs. Thereafter, 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 URL queries in the database as taint sources, too.
- (ii) Tainted VR-related PII data marking: When sensitive data enters the program from a tainted source, the data is marked as tainted data, implying that the data needs to be tracked along its propagation path. Therefore, the return values of the methods in (i) are marked as tainted data.
- (iii) Identification of taint sinks about risky access: In our approach, the tainted data may not be directly accessed with authentication but they may cause leaks if these data flow to a method that can be accessed by unauthorized users. We define these methods as taint sinks, such as `sendBroadcast()` and `sendDataMessage()`.
- (iv) Taint propagation & PII data leaks detection: After defining the taint sources, taint sinks, 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 tainted sink, such as `sendDataMessage()` and any user or app that can access it without permission, we identify a privacy leak existing.

D. 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 [54]. Recently, several studies have been conducted to analyze the privacy policies of mobile apps to identify problems and verify their reliability [55], [56], [57] despite few studies 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. On the one hand, the analysis of the privacy policy gives an idea about the current norms of the app's privacy policy. On the other hand, integrated with VR app analysis in Section III-B and PII data leaks identification in Section III-C, this module can check whether the actual code behavior is consistent with the privacy policy statement. In other words, when the privacy policy states certain types of personally identifiable information (PII) not be collected while the taint analysis reveals that the code does collect and transmit such information, then we flag this as a potential privacy issue.

We use PolicyLint [58], 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 [10]. 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. We also detect whether each privacy policy complies with the 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 [59]. We adopt a free online service called GDPRWise [60] 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. Meanwhile, we extract the apps that request permissions `HAND_TRACKING`, `FACE_TRACKING`, `BODY_TRACKING` and `EYE_TRACKING`. Moreover, 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.

TABLE III
KEYWORDS OF DANGEROUS PERMISSION

Permissions	Keywords
Account	account access, account
Bluetooth	bluetooth, bluetooth devices
Calendar	read calendar, write calendar
Contact	read contacts data, write contact, contact
Location	location, gps
Mail	mail, voicemail
Media	picture, photo, media, files, take picture, taking picture, camera
Messages	sms, receive mms, send mms, messages, read messages, read sms, send sms, mms, receive sms
Network	network, network state, wifi information, wifi, internet access, internet, network connectivity
Notification	notification, system alert window, system alert
Sensor	sensor data, sensor, fingerprint, nfc, vibrate
Biometric	hand, eye, body, face, track, biometric, iris
General Keywords	permission, access, intrusive, identity, personal info, malware, virus, malicious

E. Review Analysis

VR apps request not only conventional sensitive data (similar to mobile apps) but also biometric data with permissions to achieve an excellent user experience. Therefore, users are likely to be more concerned about the security and privacy issues caused by these data acquisitions and their impact on the user experience. We conduct a review analysis to address the users' concerns. Review analysis can help determine which privacy and security issues have the greatest impact on users, thus guiding the VR app analysis (Section III-B) and PII data leaks identification (Section III-C) to prioritize these high-risk areas. Moreover, based on review analysis, recommendations can be made for privacy policy improvements to ensure that the policy is more in line with users' expectations and actual usage.

We collect the app's user reviews from the SideQuest store and filter out non-English and malformed reviews. Thereafter, we extract the sensitive data-related reviews. In particular, we convert reviews to lowercase and consider some keywords related to dangerous permission in Android apps based on [61] to search sensitive-data-related reviews. We also consider some keywords related to biometric data requests. Table III lists all the related keywords.

IV. EXPERIMENTS AND RESULTS

In this section, we evaluate the proposed VR-SP detector by conducting comprehensive experiments on the collected 900 VR apps. We mainly consider the following seven 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 is the usage of TPL by VR apps?
RQ4: What are VR apps' major VR-platform-related security and privacy vulnerabilities?
RQ5: To what extent is PII data leaked?
RQ6: How do the VR app developers comply with the privacy policies?

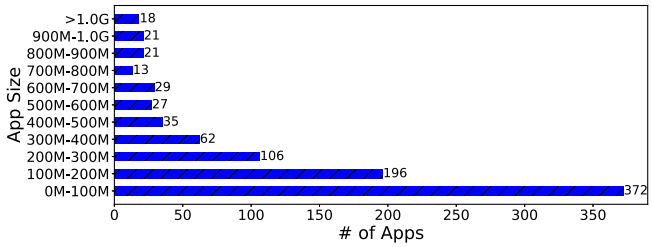


Fig. 4. App size distribution.

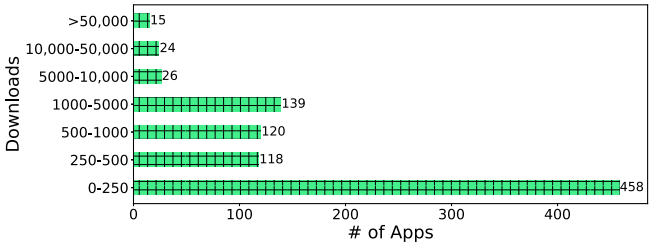


Fig. 5. Downloads distribution.

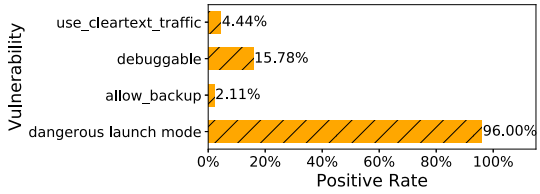


Fig. 6. Manifest analysis result.

TABLE IV
ACTIVITIES WITH DANGEROUS LAUNCH MODES

Types of Activities with Dangerous Launch Modes	Amount
com.unity3d.player.UnityPlayerActivity	775
com.epicgames.ue4.GameActivity	54
com.google Domain Activity	14
com.epicgames.unreal Domain Activity	12
com.godot.game.GodotApp Activity	3
com.oculus Domain Activity	3
com.deepinc Domain Activity	2
com.microsoft Domain Activity	2
Others	27

RQ7: How concerned are users about the security and privacy of VR apps?

Data Preparation. As mentioned in Section III-A, we collect 900 VR apps from five categories in the context of VR (the complete app list is given in our repository). Fig. 4 and Fig. 5 summarize the app size and download distribution of those 900 VR apps. It can be found that many apps are less than 100MB, which may lead 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 stages.

A. 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 Fig. 6. It can be found that 96.00% of the VR apps have activities having dangerous launch modes. 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 *uses_cleartext_tracffic* are 2.11%, 15.78% and 4.44%, respectively. Enabling *allow_backup* and *debuggable* causes a risk of coping and tampering with data from the device. This is even more dangerous in VR devices with human biometrics collected. Further, as mentioned in Section II-B1, *uses_cleartext_tracffic* may cause a MITM attack.

For these activities with dangerous launch modes, we further analyzed their specific contents and reported the results in Table IV. 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., Epic Games, 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 V 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 of security risks.

TABLE V
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.stormx.forbiddencityjourney.MainActivity
9bbd5~	com.pico.loginpaysdk.auth.TransferStationActivity com.unity3d.player.UnityPlayerActivity
a2114~	com.epicgames.ue4.GameActivity com.google.ar.core.InstallActivity

We also count the most used dangerous permissions, which have *dangerous* protection level in official Android documentation, and the number of the used Meta permissions mentioned in Section III-B. Functionally related permissions are grouped (e.g., `WRITE_EXTERNAL_STORAGE` and `READ_EXTERNAL_STORAGE` are grouped into `STORAGE`). As reported in Fig. 7, most VR apps use network info permission though some of them use permissions related to audio access. This is because the social properties and immersive nature of VR apps require network access and recording of the user's voice. Meanwhile, 99 apps use Meta 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.

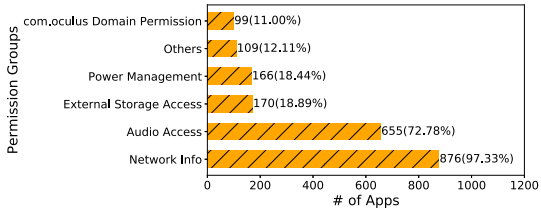


Fig. 7. Permissions at dangerous protection level.

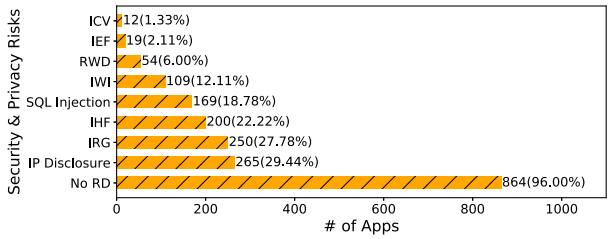


Fig. 8. Results of code analysis.

TABLE VI
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	✓	✓	✓	✓	✓	✓	✓	✓

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.

B. RQ2: What Are VR Apps' Major OS-Related Security and Privacy Vulnerabilities?

As mentioned in Section III-B, we detect security and privacy vulnerabilities from decompiled Java and Smali files. The results are shown in Fig. 8. It can be found that most VR apps have the no-root-detection vulnerability of VR devices (864 apps), thereby leading to private information leakage caused by accessing databases with root privileges. Meanwhile, 186 apps have IP disclosure risks, which may cause private information (e.g., location) to be tracked. IRG may cause private information leaks by speculating random numbers on critical functions such as `generateDefaultSessionId` and `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 [62]. Table VI 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 VII summarizes the most typical 13 kinds of trackers of VR apps. We observe that 159 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, [63] indicated that sharing sensitive data with distinct advertisers (trackers), such as *Unity3d Ads* is egregious.

TABLE VII
TRACKER ANALYSIS

Trackers	# of Apps
Google Play Billing Library / Service	96
Unity3d Ads	55
Google Firebase Analytics	20
GameAnalytics	12
Umeng Analytics	5
Umeng Common SDK logging	5
Bugsnag	5
Microsoft Domain	4
Meta (Facebook) Domain	4
Google AdMob	4
Amplitude	2
AppMetrica	2
Others	5

In particular, some specific non-VR apps also have some OS-related security and privacy vulnerabilities. Reference [19] analyzed contract tracing apps and found that most contract tracing apps had IEF and IHF. The study [64] also conducted an empirical study of Android app vulnerabilities and found that the OS-related security and privacy vulnerability that most apps had was SSL connection vulnerability (e.g., ICV and RWD). The results show that VR apps and non-VR apps have different distributions of OS-related security and privacy vulnerabilities.

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

C. RQ3: What Is the Usage of TPL by VR Apps?

As mentioned in Section III-B2, we utilize LibScan to detect the TPL distribution in VR apps. Firstly, we detect whether 205 vulnerable TPLs exist in our collected apps. The result shows that only one VR app has vulnerable TPL (TPL name: `nifi-web-contect-access-1.1.1`). As for the detection of 255 general TPLs, we find that 198 VR apps have general TPLs. The TPL distribution is shown in Table VIII. It can be found that most VR apps use `java.inject` TPL where `java.inject` is a Java library for dependency injection that provides a simple set of annotations to implement dependency injection. Dependency injection is a design pattern used to implement Inversion of Control (IoC) as a way to improve the modularity and testability of the code. As for the detection of permission in TPL, we searched permission-related APIs in the TPL packages and found that only one app uses dangerous permission APIs in the TPL. The TPL is Guava and it requests `AUDIO` and `BLUETOOTH`. The overall results show that VR apps make limited use of TPLs in traditional Android apps and do not request too many dangerous permissions in TPLs.

TABLE VIII
TPL DISTRIBUTION

TPL name	# of Apps	Package Name
javax.inject	172	`javax.inject`
annotations-java5/support-annotataions	88	`android.support.annotation`
Guava	70	`com.google.common`
Gson	44	`com.google.gson`
jsr305	33	`javax.annotation`
FasterXML-Jackson-Core	21	`com.fasterxml.jackson.core`
Dagger	8	`dagger`
slf4j-android	1	`org.slf4j.impl`

TABLE IX
INCONSISTENCY OF BIOMETRIC FUNCTION USAGE

Biometric Function	# of Apps
Hand-Tracking Function	330
Body-Tracking Function	119
Face-Tracking Function	115
Eye-Tracking Function	113

Answer to RQ3: Only one VR app adopts vulnerable TPLs. 22.00% of VR apps utilize general TPLs and only one app request dangerous permission in TPLs.

D. RQ4: 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 553 apps from 900 apps. Depending on different approaches (Mono-based and IL2CPP-based), we obtain 266 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 37 apps adopting the Unity IAP function. According to the taint analysis, there exist IAP no-verification vulnerabilities in these 37 apps. Moreover, Table IX shows that there are 348 apps adopting biometric data collection functions while having no permission requests in the manifest file. Specifically, 330 apps use the hand-tracking function while indicating no request for that permission in the `Androidmanifest.xml` file. Among them, 119 apps exploit body-tracking functions, 115 apps invoke face-tracking functions, and 113 apps call eye-tracking functions. This implies that *a significant number of apps do not adhere to the specifications of the Meta 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.

Notably, IAP vulnerabilities also exist in non-VR mobile apps, but most of them do not adopt Unity API to achieve this function. Thus, they have no Unity IAP vulnerabilities. The study [65] found that 37% Android apps with at least 100,000 users embed third-party payment functionality. Hundreds of them violated security rules and contained various potential security risks, allowing an attacker to consume almost every aspect of commodities or services in life without actually purchasing them or deceiving others to pay for them. The findings show that non-VR apps also have similar security risks in in-app payment although they do not adopt Unity API. Our approach mainly focuses on Unity IAP vulnerabilities in VR apps since most VR apps are developed by Unity. Moreover, the inconsistent usage of biometric functions does not exist in non-VR apps since conventional non-VR apps do not collect biometric data.

[2]This issue was raised in the Meta community forum, but no explicit answer has been given until January 2025: https://communityforums.atmeta.com/t5/Oculus-Quest-2-and-Quest/Unity-Oculus-Integration-bug-Hand-tracking-always-enabled-no/td-p/753132

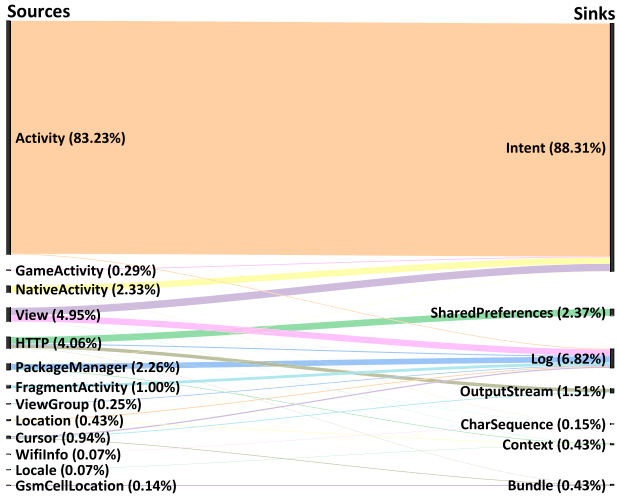


Fig. 9. Data leaks detection by data flow analysis.

Answer to RQ4: Although only 4.11% of VR apps have used Unity IAP functions, all of them have IAP no-verification vulnerabilities. 38.67% of Unity VR apps have inconsistency between permission requests and biometric function usage, thereby causing leakage risks of human biometrics.

E. RQ5: To What Extent Is PII Data Leaked?

We also adopt a taint analysis of PII data leaks. Fig. 9 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 sources for obtaining PII data. The largest amount of PII data flows 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 [66].

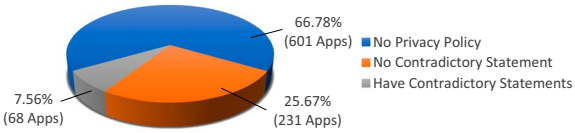


Fig. 10. App privacy policy distribution.

TABLE X
GDPR COMPLIANCE CHECK

GDPR Violation Term	Risk Level	# of Apps
Missing Social Media Clause	High	83
Missing Data Sharing Information	High	142
Missing GDPR Roles	High	184
Missing Data Subject Rights	High	94
Missing Legal Ground	High	181
Missing Data Retention Information	High	151
Missing Timestamp	High	70
Missing National Authority	High	151
Missing Sections	High	105
Missing GDPR Specificity	High	105
Missing Dedicated Privacy Mailbox	Medium	65
Missing Data Security Information	Medium	91
Missing Top Level Link	Medium	102
No Violation	N/A	53

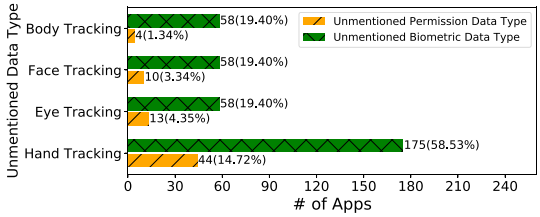
Answer to RQ5: A number of VR apps have the leakage risks of PII sensitive data. Most data flow from activity-related methods to intent-related methods.

F. RQ6: How Do the VR App Developers Comply With the Privacy Policies?

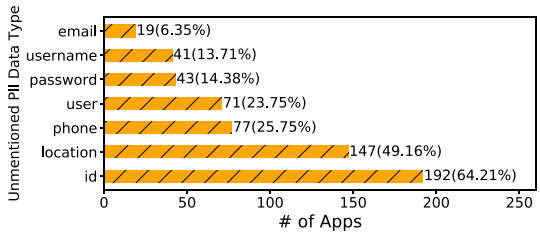
As mentioned in Section III-D, we collect policy statements from apps' privacy policies based on predefined ontologies by PolicyLint [58]. We check whether there are contradictory statements and GPDR violations in privacy policies and whether they are consistent with the app analysis results.

As shown in Fig. 10, unexpectedly 601 apps (66.78%) have no privacy policy though only 299 apps (33.22%) have privacy policies. Among 299 apps with privacy policies, 68 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 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 X reports the GDPR compliance check results. We find that only 53 apps among these 299 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., 184) miss GDPR roles. 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.



(a) Permission & Biometrics Inconsistency Check

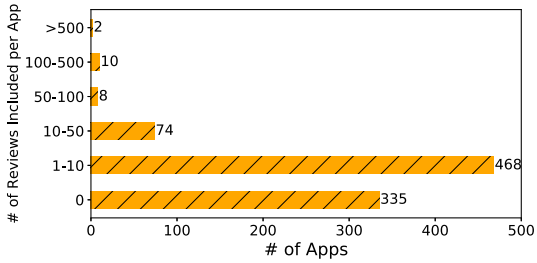


(b) PII Inconsistency Check

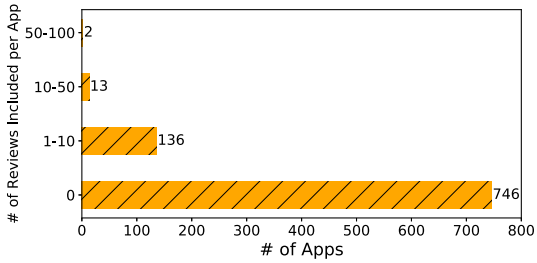
Fig. 11. Privacy policy inconsistency check (% indicates proportion of apps with privacy policy.)

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 (Section III-D). Integrated with VR app analysis and PII data leaks identification, 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 Fig. 11, we find that 52 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, 44 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 Fig. 11, we find that most apps with privacy policies (243 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, 192 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 Fig. 11. In addition, 175 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 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 RQ6: Less than 40% of VR apps offer privacy policies though 22.74% of them contain contradictory statements. Meanwhile, 17.73% of 299 VR apps with privacy policies comply with GDPR regulations, 81.27% of them have no explicit mention of PII data usage in detail, and about 60% have inconsistency between human biometrics collection and privacy policy.



(a) Distribution of All Reivews



(b) Distribution of Sensitive-Data-Related Reviews

Fig. 12. Review Distribution

G. RQ7: How Concerned Are Users About the Security and Privacy of VR Apps?

As mentioned in Section III-E, we collect user reviews from the SideQuest app store to analyze how users are concerned about the security and privacy of VR apps. Out of 900 apps, we find 897 apps with user reviews in SideQuest. After filtering out non-English reviews and malformed reviews, we collect 7,772 reviews in total. The distribution is shown in Fig. 12. The result shows that 803 VR apps have fewer than 10 reviews. By using a keyword search to extract sensitive-data-related reviews, we find that there are only 730 reviews. The distribution is shown in Fig. 12. Most VR apps have fewer than 10 related reviews. This indicates that users may not be quite concerned about the use of sensitive data. Table XI reports the distribution of different types of sensitive-data-related reviews based on different permissions. We find that most reviews focus on biometric data. This suggests that the use of VR-specific biometric data still receives much attention despite the limited number of user reviews. It also indicates that the privacy policy should focus more on the collection and usage of biometric data. The review analysis also provides the corroboration of the VR app analysis and the PII data analysis.

Answer to RQ7: 9.39% of VR apps user review mention sensitive-data-related information. Most VR apps have less than 10 reviews. 58.63% of sensitive-data-related reviews mention biometric data.

H. 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. As for TPL usage, this app has no vulnerable TPLs but adopts some general TPLs such as `FasterXML-Jackson-Core`, `jsr305` and `java.inject` but does not request any dangerous permission in TPLs. 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 but are adopted in the source code. In addition, the privacy policy contains GDPR violations, such as missing data sharing information and missing top-level links. It includes 11 reviews, two of which mention sensitive-data-related information.

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. Moreover, 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. It also has no vulnerable TPLs but utilizes general TPLs such

TABLE XI
DIFFERENT TYPES OF SENSITIVE-DATA-RELATED REVIEWS

Permissions	# of Reviews
Account	53
Bluetooth	2
Calendar	1
Contact	9
Location	167
Mail	22
Media	100
Messages	24
Network	32
Notification	5
Sensor	4
Biometric	428
General	79

as `Gson`, `support-annotataions` and `javax.inject`. All utilized TPLs do not request dangerous permissions. 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`, and `phone` data collection are not mentioned in the privacy policy but they are adopted in the source code. As for review analysis, this app includes 47 reviews, seven of which mention sensitive data-related information.

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 `uses_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 [67]. 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 [68]) 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: Use TPLs cautiously during VR app development.

Despite only a limited amount of apps using vulnerable TPLs and TPLs requesting dangerous permissions, developers need to carefully adopt TPLs. Developers need to check the security of TPLs to ensure that there are no known vulnerabilities, as well as regularly reviewing and updating libraries to guard against security risks.

Advice 6: 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 7: 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.

Advice 8: Create privacy policies considering users' concerns.

According to the result of the review analysis, we find that some reviews are concerned with sensitive data. The privacy policy creators should clarify the data collection and usage so as to ensure that policies are in line with user expectations and actual utilization.

V. 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 are not open-sourced in the Meta 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 [69], libGDX [70], and so on. Moreover, we only analyze VR apps working on Meta 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 900 VR apps, which are almost 5 $\times$ of the state-of-the-art tool OVRSEEN [10], 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 refer to predefined patterns based on [19], 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 check 20% of the identified paths and find no false positives, thus mitigating its side effect. With respect to another internal validity threat caused by the accuracy of the GDPRWise, we manually sample 50 privacy policy cases to verify the accuracy of their GDPR compliance detection and find 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.

VI. 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. [71] proposed a static tool to analyze inter-communication between Android Java and JavaScript codes. Sandeep [72] combined deep learning and static analysis to detect Android malware with high accuracy. Regarding dynamic analysis, Reardon et al. [73] constructed a testing environment to detect whether the app bypassed the permission model to access protected data. Huang et al. [74] 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 [75]. For example, Trimananda et al. [10] proposed a method, namely OVRSEEN to analyze the privacy policies in Meta 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 [76]. Yarramreddy et al. [77] proposed a forensic analysis of VR social apps to reveal some forensically relevant data from network traffic and the VR systems. Casey et al. [9] 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. [78] 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. [79] 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. [13] 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.

Comparison with our preliminary study. Compared with our preliminary study presented in [11], we have made three substantial contributions to this article.

- In addition to what was analyzed in [11], considering that VR apps also have an ecology of TPL usage similar to Android apps, we add a module for TPL analysis in the VR-SP detector. Specifically, we analyze the distribution of TPLs and the usage of vulnerable TPLs.
- To understand users' concerns about app security and privacy issues, we add a module for analyzing user reviews to the VR-SP detector. Using this module we collected 7,772 comments and used keyword search methods to deeply analyze user perceptions of security, privacy, and the usage of biometric data.
- We significantly scale the number of detected apps of our VR-SP detector from 500 to 900, thereby achieving a more comprehensive coverage of app types.

VII. 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, privacy policy analysis methods, and review analysis methods. Using the VR-SP detector, we have conducted the security and privacy assessment of 900 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.

In the future, we will adopt dynamic testing to address the limitations caused by static analysis and shelling. Therefore, we can achieve a detailed security and privacy assessment. We will also integrate more tools to analyze VR apps running on top of diverse VR devices and VR apps developed by different languages (e.g., JavaScript).

References

[1] I. Wohlgenannt, A. Simons, and S. Stieglitz, "Virtual reality," *Business & Information Systems Engineering*, vol. 62, no. 5, pp. 455–461, Oct. 2020.
[2] F. B. Insights, "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." [Online]. Available: https://www.fortunebusinessinsights.com/industry-reports/virtual-reality-market-101378
[3] S. Mystakidis, "Metaverse," *Encyclopedia*, vol. 2, no. 1, pp. 486–497, 2022. [Online]. Available: https://www.mdpi.com/2673-8392/2/1/31
[4] A. Christopher Santoso and P. Santoso, "Aplikasi ruangan maya berbasis Android OS pada headset virtual reality Oculus Quest 2," *Jurnal FORTECH*, vol. 3, no. 2, pp. 51–56, Sep. 2022. [Online]. Available: https://journal.fortei7.org/index.php/fortech/article/view/357
[5] J. Hui, Y. Zhou, M. Oubibi, W. Di, L. Zhang, and S. Zhang, "Research on art teaching practice supported by virtual reality (VR) technology in the primary schools," *Sustainability*, vol. 14, no. 3, 2022, Art. no. 1246. [Online]. Available: https://www.mdpi.com/2071-1050/14/3/1246
[6] Y. Huang, Y. J. Li, and Z. Cai, "Security and privacy in metaverse: A comprehensive survey," *Big Data Mining Analytics*, vol. 6, no. 2, pp. 234–247, 2023.
[7] T. Mustafa, R. Matovu, A. Serwadda, and N. Muirhead, "Unsure how to authenticate on your VR headset? Come on, use your head!" in *Proc. 4th ACM Int. Workshop Secur. Privacy Analytics (IWSPA)*, New York, NY, USA: ACM, 2018, pp. 23–30, doi: 10.1145/3180445.3180450.
[8] M. Vondráček, I. Baggili, P. Casey, and M. Mekni, "Rise of the metaverse's immersive virtual reality malware and the man-in-the-room attack & defenses," *Comput. & Secur.*, vol. 127, 2023, Art. no. 102923. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822003157
[9] P. Casey, I. Baggili, and A. Yarramreddy, "Immersive virtual reality attacks and the human joystick," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 2, pp. 550–562, Mar./Apr. 2021.

[10] R. Trimananda, H. Le, H. Cui, J. T. Ho, A. Shuba, and A. Markopoulou, "OVRseen: Auditing network traffic and privacy policies in Oculus VR," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur. 22),* Boston, MA, USA: USENIX Association, Aug. 2022, pp. 3789–3806. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/trimananda

[11] H. Guo, H.-N. Dai, X. Luo, Z. Zheng, G. Xu, and F. He, "An empirical study on Oculus virtual reality applications: Security and privacy perspectives," in *Proc. IEEE/ACM 46th Int. Conf. Softw. Eng. (ICSE)*, New York, NY, USA: ACM, 2024, doi: 10.1145/3597503.3639082.

[12] C. Cortés, P. Pérez, and N. García, "Unity3d-based app for 360VR subjective quality assessment with customizable questionnaires," in *Proc. IEEE 9th Int. Conf. Consum. Electron. (ICCE-Berlin)*, 2019, pp. 281–282.

[13] C. Zuo and Z. Lin, "Playing without paying: Detecting vulnerable payment verification in native binaries of Unity mobile games," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.),* Boston, MA, USA: USENIX Association, Aug. 2022, pp. 3093–3110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/zuo

[14] R. Cheng, N. Wu, S. Chen, and B. Han, "Reality check of metaverse: A first look at commercial social virtual reality platforms," in *Proc. IEEE Conf. Virtual Reality 3D User Interfaces Abstr. Workshops (VRW)*, 2022, pp. 141–148.

[15] S.-M. Park and Y.-G. Kim, "A metaverse: Taxonomy, components, applications, and open challenges," *IEEE Access*, vol. 10, pp. 4209–4251, 2022.

[16] T. Huynh-The, Q.-V. Pham, X.-Q. Pham, T. T. Nguyen, Z. Han, and D.-S. Kim, "Artificial intelligence for the metaverse: A survey," *Eng. Appl. Artif. Intell.*, vol. 117, 2023, Art. no. 105581. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197622005711

[17] J. Lee, "A study on the intention and experience of using the metaverse," *Jahr: Europski časopis za bioetiku*, vol. 13, no. 1, pp. 177–192, 2022.

[18] H. Ning et al., "A survey on metaverse: The state-of-the-art, technologies, applications, and challenges," 2021, *arXiv:2111.09673*.

[19] R. Sun, W. Wang, M. Xue, G. Tyson, S. Camtepe, and D. C. Ranasinghe, "An empirical assessment of global COVID-19 contact tracing applications," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, 2021, pp. 1085–1097.

[20] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[21] G. Bai et al., "All your sessions are belong to us: Investigating authenticator leakage through backup channels on android," in *Proc. 20th Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, 2015, pp. 60–69.

[22] A. K. Jha, S. Lee, and W. J. Lee, "Developer mistakes in writing android manifests: An empirical study of configuration errors," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR)*, 2017, pp. 25–36.

[23] A. Possemato and Y. Fratantonio, "Towards HTTPS everywhere on android: We are not there yet," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.),* Boston, MA, USA: USENIX Association, Aug. 2020, pp. 343–360. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/possemato

[24] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Commun. Surveys & Tut.*, vol. 18, no. 3, pp. 2027–2051, 3rd Quart. 2016.

[25] G. LaMalva and S. Schmeelk, "MOBSF: Mobile health care Android applications through the lens of open source static analysis," in *Proc. IEEE MIT Undergraduate Res. Technol. Conf. (URTC)*, 2020, pp. 1–4.

[26] H. Darvish and M. Husain, "Security analysis of mobile money applications on Android," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2018, pp. 3072–3078.

[27] F. H. Shezan, S. F. Afroze, and A. Iqbal, "Vulnerability detection in recent Android apps: An empirical study," in *Proc. Int. Conf. Netw., Syst. Secur. (NSysS)*, 2017, pp. 55–63.

[28] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL injection attack using machine learning techniques: A systematic literature review," *J. Cybersecur. Privacy*, vol. 2, no. 4, pp. 764–777, 2022. [Online]. Available: https://www.mdpi.com/2624-800X/2/4/39

[29] M. Oltrogge, N. Huaman, S. Amft, Y. Acar, M. Backes, and S. Fahl, "Why Eve and Mallory still love Android: Revisiting TLS (in) security in Android applications," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.),* Boston, MA, USA: USENIX Association, Aug. 2021, pp. 4347–4364. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/oltrogge

[30] J. P. Hughes and W. Diffie, "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*, vol. 20, no. 3, pp. 18–40, Jul. 2022, doi: 10.1145/3546933.

[31] A. R. Sai, J. Buckley, and A. Le Gear, "Privacy and security analysis of cryptocurrency mobile applications," in *Proc. 5th Conf. Mobile Secure Services (MobiSecServ)*, 2019, pp. 1–6.

[32] D. J. Leith and S. Farrell, "Contact tracing app privacy: What data is shared by Europe's gaen contact tracing apps," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–10.

[33] V. Ang and L. K. Shar, "COVID-19 one year on – security and privacy review of contact tracing mobile apps," *IEEE Pervasive Comput.*, vol. 20, no. 4, pp. 61–70, Oct./Dec. 2021.

[34] S. Shuai, D. Guowei, G. Tao, Y. Tianchang, and S. Chenjie, "Modelling analysis and auto-detection of cryptographic misuse in Android applications," in *Proc. IEEE 12th Int. Conf. Dependable, Autonomic Secure Comput.*, 2014, pp. 75–80.

[35] S. Yoo and X. Chen, "Secure keyed hashing on programmable switches," in *Proc. ACM SIGCOMM 2021 Workshop Secure Programmable Netw. INfrastructure (SPIN)*, New York, NY, USA: ACM, 2021, pp. 16–22, doi: 10.1145/3472873.3472881.

[36] H. Zhang, D. She, and Z. Qian, "Android root and its providers: A double-edged sword," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2015, pp. 1093–1104, doi: 10.1145/2810103.2813714.

[37] K. Kollnig et al., "A fait accompli? An empirical study into the absence of consent to Third-Party tracking in Android apps," in *Proc. 17th Symp. Usable Privacy Secur. (SOUPS),* Boston, MA, USA: USENIX Association, Aug. 2021, pp. 181–196. [Online]. Available: https://www.usenix.org/conference/soups2021/presentation/kollnig

[38] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: fast and accurate detection of third-party libraries in Android apps," in *Proc. 38th Int. Conf. Softw. Eng. Companion (ICSE)*, New York, NY, USA: ACM, 2016, pp. 653–656, doi: 10.1145/2889160.2889178.

[39] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in Android and its security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2016, pp. 356–367, doi: 10.1145/2976749.2978333

[40] X. Zhan et al., "Automated third-party library detection for Android applications: Are we there yet?" in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, 2020, pp. 919–930.

[41] U. Technologies, "Unity documentation - 2D or 3D projects." 2023. [Online]. Available: https://docs.unity3d.com/

[42] I. Epic Games, "Unreal Engine." 2023. [Online]. Available: https://www.unrealengine.com/

[43] A. Desnos and G. Gueguen, "Androguard documentation." [Online]. Available: https://androguard.readthedocs.io/en/latest/

[44] X. Yang and X. Zhang, "A study of user privacy in Android mobile AR apps," in *Proc. 37th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, New York, NY, USA: ACM, 2023, doi: 10.1145/3551349.3560512.

[45] C. Ren, Y. Zhang, H. Xue, T. Wei, and P. Liu, "Towards discovering and understanding task hijacking in Android," in *Proc. 24th USENIX Conf. Secur. Symp. (SEC),* Boston, MA, USA: USENIX Association, 2015, pp. 945–959.

[46] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl, "Protecting software through obfuscation: Can it keep pace with progress in code analysis?" *ACM Comput. Surv.*, vol. 49, no. 1, pp. 1–37, Apr. 2016, doi: 10.1145/2886012.

[47] I. F. Elashry, O. S. Farag Allah, A. M. Abbas, and S. El-Rabaie, "A new diffusion mechanism for data encryption in the ECB mode," in *Proc. Int. Conf. Comput. Eng. & Syst.*, 2009, pp. 288–293.

[48] M. Li et al.,"LibD: Scalable and precise third-party library detection in Android markets," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, 2017, pp. 335–346.

[49] Y. Wu, C. Sun, D. Zeng, G. Tan, S. Ma, and P. Wang, "LibScan: Towards more precise third-party library identification for Android applications," in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur. 23),* Anaheim, CA. Boston, MA, USA: USENIX Association, Aug. 2023, pp. 3385–3402. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/wu-yafei

[50] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2012, pp. 217–228, doi: 10.1145/2382196.2382222.

[51] F. Nusrat, F. Hassan, H. Zhong, and X. Wang, "How developers optimize virtual reality applications: A study of optimization commits in open

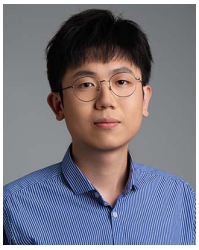
source Unity projects," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, 2021, pp. 473–485.
[52] "dnSpy." 2020. [Online]. Available: https://github.com/dnSpy/dnSpy
[53] S. Arzt et al., "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proc. 35th ACM SIGPLAN Conf. Program. Lang. Des. Implementation (PLDI)*, New York, NY, USA: ACM, 2014, pp. 259–269, doi: 10.1145/2594291.2594299.
[54] C. Chang, H. Li, Y. Zhang, S. Du, H. Cao, and H. Zhu, "Automated and personalized privacy policy extraction under GDPR consideration," in *Proc. Wireless Algorithms Syst., Appl.*, E. S. Biagioni, Y. Zheng, and S. Cheng, Eds., Cham, Switzerland: Springer International Publishing, 2019, pp. 43–54.
[55] L. Verderame, D. Caputo, A. Romdhana, and A. Merlo, "On the (un)reliability of privacy policies in Android apps," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2020, pp. 1–9.
[56] J. Benjumea, J. Ropero, O. Rivera-Romero, E. Dorronzoro-Zubiete, and A. Carrasco, "Assessment of the fairness of privacy policies of mobile health apps: Scale development and evaluation in cancer apps," *JMIR Mhealth Uhealth*, vol. 8, no. 7, Jul. 2020, Art. no. e17134, doi: 10.2196/17134.
[57] S. Liao, C. Wilson, L. Cheng, H. Hu, and H. Deng, "Measuring the effectiveness of privacy policies for voice assistant applications," in *Annu. Comput. Secur. Appl. Conf. (ACSAC)*, New York, NY, USA: ACM, 2020, pp. 856–869, doi: 10.1145/3427228.3427250.
[58] B. Andow et al., "PolicyLint: Investigating internal privacy policy contradictions on Google Play," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.),* Santa Clara, CA. Boston, MA, USA: USENIX Association, Aug. 2019, pp. 585–602. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/andow
[59] H. Li, L. Yu, and W. He, "The impact of GDPR on global technology development," *J. Global Inf. Technol. Manage.*, vol. 22, no. 1, pp. 1–6, 2019, doi: 10.1080/1097198X.2019.1569186.
[60] G. BV. "GDPRWise policy checker." 2023. [Online]. Available: https://gdprwise.eu/policy-checker/
[61] D. C. Nguyen, E. Derr, M. Backes, and S. Bugiel, "Short text, large effect: Measuring the impact of user reviews on Android app security & privacy," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2019, pp. 555–569.
[62] Y. Wu et al., "Privacy leakage via unrestricted motion-position sensors in the age of virtual reality: A study of snooping typed input on virtual keyboards," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2023, pp. 3382–3398.
[63] B. Andow et al., "Actions speak louder than words: Entity-Sensitive privacy policy and data flow analysis with PoliCheck," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, Boston, MA, USA: USENIX Association, Aug. 2020, pp. 985–1002. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/andow
[64] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, "Understanding the evolution of android app vulnerabilities," *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 212–230, Mar. 2021.
[65] W. Yang et al., "Show me the money! finding flawed implementations of third-party in-app payment in android apps." in *Proc. Netw. Distrib. System Secur. (NDSS) Symp.*, 2017, p. 15.
[66] H. Zhou et al., "Demystifying diehard Android apps," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE),* New York, NY, USA: ACM, 2021, pp. 187–198, doi: 10.1145/3324884.3416637.
[67] S.-T. Sun, A. Cuadros, and K. Beznosov, "Android rooting: Methods, detection, and evasion," in *Proc. 5th Annu. ACM CCS Workshop Secur. Privacy Smartphones Mobile Devices (SPSM),* New York, NY, USA: ACM, 2015, pp. 3–14, doi: 10.1145/2808117.2808126.
[68] A. W. Appel, "Verification of a cryptographic primitive: Sha-256," *ACM Trans. Program. Lang. Syst.*, vol. 37, no. 2, pp. 1–31, Apr. 2015, doi: 10.1145/2701415.
[69] W. Qiu and A. Yuille, "Unrealcv: Connecting computer vision to unreal engine," in *Proc. Comput. Vis. ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds. Cham, Switzerland: Springer International Publishing, 2016, pp. 909–916.
[70] L. Stemkoski, *The LibGDX Framework*. Berkeley, CA, USA: Apress, 2015, pp. 13–46, doi: 10.1007/978-1-4842-1500-5_2.
[71] S. Lee, J. Dolby, and S. Ryu, "HybriDroid: Static analysis framework for Android hybrid applications," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE),* New York, NY, USA: ACM, 2016, pp. 250–261, doi: 10.1145/2970276.2970368.
[72] S. HR, "Static analysis of Android malware detection using deep learning," in *Proc. Int. Conf. Intell. Comput. Control Syst. (ICCS)*, 2019, pp. 841–845.
[73] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur. 19)*. Santa Clara, CA. Boston, MA, USA: USENIX Association, Aug. 2019, pp. 603–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/reardon
[74] X. Huang, A. Zhou, P. Jia, L. Liu, and L. Liu, "Fuzzing the android applications with http/https network data," *IEEE Access*, vol. 7, pp. 59951–59962, 2019.
[75] S. Li et al., "An exploratory study of bugs in extended reality applications on the web," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2020, pp. 172–183.
[76] D. Devecsery, P. M. Chen, J. Flinn, and S. Narayanasamy, "Optimistic hybrid analysis: Accelerating dynamic analysis through predicated static analysis," *SIGPLAN Not.*, vol. 53, no. 2, pp. 348–362, Mar. 2018, doi: 10.1145/3296957.3177153.
[77] A. Yarramreddy, P. Gromkowski, and I. Baggili, "Forensic analysis of immersive virtual reality social applications: A primary account," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, 2018, pp. 186–196.
[78] J. Shim, K. Lim, S.-j. Cho, S. Han, and M. Park, "Static and dynamic analysis of Android malware and goodware written with Unity framework," *Secur. Commun. Netw.*, vol. 2018, no. 1, p. 12, Jan. 2018.
[79] S. Volokh and W. G. Halfond, "Static analysis for automated identification of valid game actions during exploration," in *Proc. 17th Int. Conf. Foundations Digit. Games (FDG)*, New York, NY, USA: ACM, 2022, doi: 10.1145/3555858.3555898.

Hanyang Guo is currently working toward the Ph.D. degree with the School of Software Engineering, Sun Yat-sen University. He is also a visiting student with the Department of Computer Science, Hong Kong Baptist University. His research interests include VR/AR software reliability and AI for software engineering.

Hong-Ning Dai (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Department of Computer Science and Engineering, The Chinese University of Hong Kong. Currently, he is an Associate Professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. His research interests include the Internet of Things, blockchain, and extended reality (XR) technologies. He has published more than 250 papers in top-tier journals and conferences with 23 000+ citations. He has served as an Associate Editor for IEEE COMMUNICATIONS SURVEY AND TUTORIALS, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON INDUSTRIAL CYBER-PHYSICAL SYSTEMS, and *Ad Hoc Networks*. He is also a Senior Member of the Association for Computing Machinery (ACM).

Xiapu Luo (Senior Member, IEEE) is a Professor with the Department of Computing, The Hong Kong Polytechnic University. His research focuses on blockchain and smart contracts security, mobile and IoT security, network security and privacy, and software engineering with papers published in top-tier security, software engineering, and networking conferences and journals. His research led to more than ten Best/Distinguished Paper Awards, including the ACM CCS'24 Distinguished Paper Award, the four ACM SIGSOFT Distinguished Paper Awards, Best DeFi Papers Award 2023, the Best Paper Award in INFOCOM'18, the Best Research Paper Award in ISSRE'16, etc. and several awards from the industry. He received the BOCHK Science and Technology Innovation Prize (FinTech)

2023 for his contribution to blockchain security. He is an ACM Distinguished Member for his research in safeguarding blockchain and smart contracts along with Android and its applications.



Gengyang Xu is currently working toward the bachelor's degree with the Department of Computer Science, Hong Kong Baptist University. His research interests include intelligent software engineering, virtual reality, and software security.

Fengliang He is currently working toward the Ph.D. degree with the Department of Computer Science, Hong Kong Baptist University. His research interest includes VR/AR system security.

Zibin Zheng (Fellow, IEEE) is currently a Professor and the Dean of the School of Software Engineering, Sun Yat-sen University, Zhuhai, China. He authored or co-authored more than 200 international journal and conference papers, including one ESI hot paper and ten ESI highly cited papers. According to Google Scholar, his papers have more than 44 000 citations. His research interests include blockchain, software engineering, and services computing. He was the BlockSys'19 and CollaborateCom16 General Co-Chair, SC2'19, ICIOT18, and IoV14 PC Co-Chair. He is a Fellow of the IET. He was the recipient of several awards, including the Top 50 Influential Papers in Blockchain of 2018, the ACM SIGSOFT Distinguished Paper Award at ICSE2010, and the Best Student Paper Award at ICWS2010.