OVRSEEN: Auditing Network Traffic and Privacy Policies in Oculus VR

Rahmadi Trimananda,[1] Hieu Le,[1] Hao Cui,[1] Janice Tran Ho,[1] Anastasia Shuba,[2] and Athina Markopoulou[1]

[1]University of California, Irvine
[2]Independent Researcher

Abstract

Virtual reality (VR) is an emerging technology that enables new applications but also introduces privacy risks. In this paper, we focus on Oculus VR (OVR), the leading platform in the VR space and we provide the first comprehensive analysis of personal data exposed by OVR apps and the platform itself, from a combined networking and privacy policy perspective. We experimented with the Quest 2 headset and tested the most popular VR apps available on the official Oculus and the SideQuest app stores. We developed OVRSEEN, a methodology and system for collecting, analyzing, and comparing network traffic and privacy policies on OVR. On the networking side, we captured and decrypted network traffic of VR apps, which was previously not possible on OVR, and we extracted data flows, defined as ⟨*app, data type, destination*⟩. Compared to the mobile and other app ecosystems, we found OVR to be more centralized and driven by tracking and analytics, rather than by third-party advertising. We show that the data types exposed by VR apps include personally identifiable information (PII), device information that can be used for fingerprinting, and VR-specific data types. By comparing the data flows found in the network traffic with statements made in the apps' privacy policies, we found that approximately 70% of OVR data flows were not properly disclosed. Furthermore, we extracted additional context from the privacy policies, and we observed that 69% of the data flows were used for purposes unrelated to the core functionality of apps.

1 Introduction

Virtual reality (VR) technology has created an emerging market: VR hardware and software revenues are projected to increase from $800 million in 2018 to $5.5 billion in 2023 [53]. Among VR platforms, Oculus VR (OVR) is a pioneering, and arguably the most popular one: within six months since October 2020, an estimated five million Quest 2 headsets were sold [16, 22]. VR technology enables a number of applications, including recreational games, physical training, health therapy, and many others [52].

VR also introduces privacy risks: some are similar to those on other Internet-based platforms (*e.g.,* mobile phones [12,13], IoT devices [3, 17], and Smart TVs [37, 67]), but others are unique to VR. For example, VR headsets and hand controllers are equipped with sensors that may collect data about the user's physical movement, body characteristics and activity, voice activity, hand tracking, eye tracking, facial expressions, and play area [27, 36], which may in turn reveal information about our physique, emotions, and home. The privacy aspects of VR platforms are currently not well understood [2].

To the best of our knowledge, our work is the first large scale, comprehensive measurement and characterization of privacy aspects of OVR apps and platform, from a combined network and privacy policy point of view. We set out to characterize how sensitive information is collected and shared in the VR ecosystem, in theory (as described in the privacy policies) and in practice (as exhibited in the network traffic generated by VR apps). We center our analysis around the concept of *data flow*, which we define as the tuple ⟨*app, data type, destination*⟩ extracted from the network traffic. First, we are interested in the sender of information, namely the VR *app*. Second, we are interested in the exposed *data types*, including personally identifiable information (PII), device information that can be used for fingerprinting, and VR sensor data. Third, we are interested in the recipient of the information, namely the *destination* domain, which we further categorize into entity or organization, first *vs.* third party *w.r.t.* the sending app, and ads and tracking services (ATS). Inspired by the framework of contextual integrity [40], we also seek to characterize whether the data flows are appropriate or not within their context. More specifically, our notion of context includes: *consistency*, *i.e.,* whether actual data flows extracted from network traffic agree with the corresponding statements made in the privacy policy; *purpose*, extracted from privacy policies and confirmed by destination domains (*e.g.,* whether they are ATS); and other information (*e.g.,* "notice and consent"). Our methodology and system, OVRSEEN, is depicted on Fig. 1. Next we summarize our methodology and findings.

Network traffic: methodology and findings. We were able to explore 140 popular, paid and free, OVR apps; and then capture, decrypt, and analyze the network traffic they generate in order to assess their practices with respect to collection and sharing of personal data on the OVR platform.

OVRSEEN collects network traffic without rooting the Quest 2, by building on the open-source AntMonitor [54], which we had to modify to work on the Android 10-based Ocu-

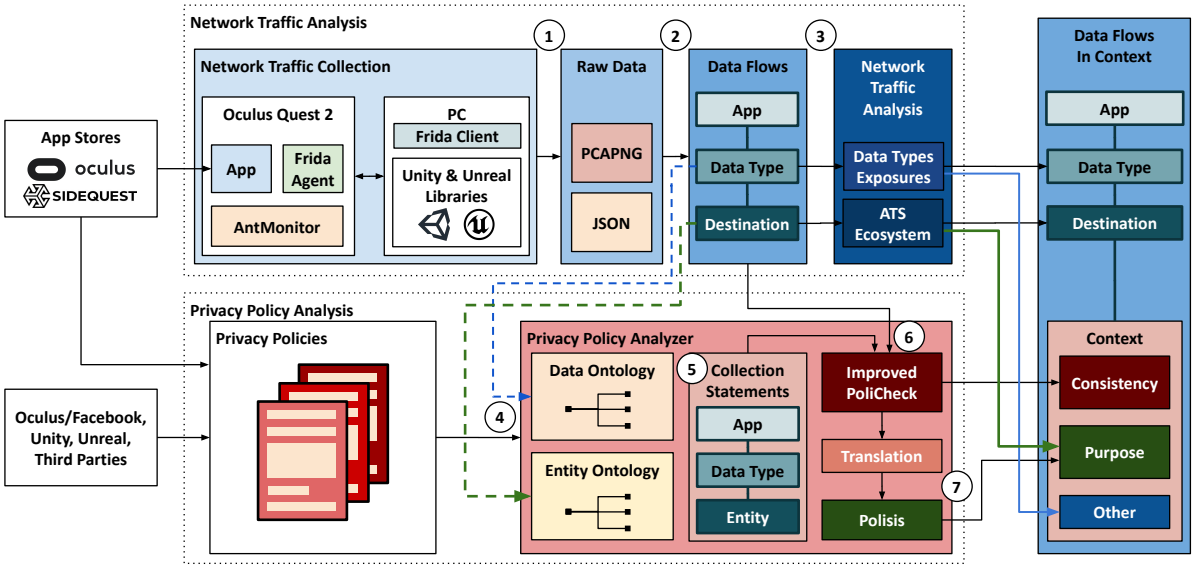


Figure 1: **Overview of OVRSEEN.** We select the most popular apps from the official Oculus and SideQuest app stores. First, we experiment with them and analyze their **network traffic:** (1) we obtain raw data in PCAPNG and JSON; (2) we extract data flows ⟨*app, data type, destination*⟩; and (3) we analyze them *w.r.t.* data types and ATS ecosystem. Second, we analyze the same apps' (and their used libraries') **privacy policies**: (4) we build VR-specific data and entity ontologies, informed both by network traffic and privacy policy text; and (5) we extract collection statements ⟨*app, data type, entity*⟩ from the privacy policy. Third, we **compare the two**: (6) using our improved PoliCheck, we map each data flow to a collection statement, and we perform network-to-policy consistency analysis. Finally, (7) we translate the sentence containing the collection statement into a text segment that Polisis can use to extract the data collection purpose. The end result is that data flows, extracted from network traffic, are augmented with additional **context**, such as consistency with policy and purpose of collection.

lus OS. Furthermore, we successfully addressed new technical challenges for decrypting network traffic on OVR. OVRSEEN combines dynamic analysis (using Frida [45]) with binary analysis to find and bypass certificate validation functions, even when the app contains a stripped binary [66]. This was a challenge specific to OVR: prior work on decrypting network traffic on Android [37,55] hooked into standard Android SDK functions and not the ones packaged with Unity and Unreal, which are the basis for game apps.

We extracted and analyzed data flows found in the collected network traffic from the 140 OVR apps, and we made the following observations. We studied a broad range of 21 data types that are exposed and found that 33 and 70 apps send PII data types (*e.g.,* Device ID, User ID, and Android ID) to first-and third-party destinations, respectively (see Table 3). Notably, 58 apps expose VR sensory data (*e.g.,* physical movement, play area) to third-parties. We used state-of-the-art blocklists to identify ATS and discovered that, unlike other popular platforms (*e.g.,* Android and Smart TVs), OVR exposes data primarily to tracking and analytics services, and has a less diverse tracking ecosystem. Notably, the blocklists identified only 36% of these exposures. On the other hand, we found no data exposure to advertising services as ads on OVR is still in an experimental phase [44].

Privacy policy: methodology and findings. We provide an NLP-based methodology for analyzing the privacy policies that accompany VR apps. More specifically, OVRSEEN maps each data flow (found in the network traffic) to its corresponding data collection statement (found in the text of the privacy policy), and checks the *consistency* of the two. Furthermore, it extracts the *purpose* of data flows from the privacy policy, as well as from the ATS analysis of destination domains. Consistency, purpose, and additional information provide *context*, in which we can holistically assess the appropriateness of a data flow [40]. Our methodology builds on, combines, and improves state-of-the-art tools originally developed for mobile apps: PolicyLint [4], PoliCheck [5], and Polisis [21]. We curated VR-specific ontologies for data types and entities, guided by both the network traffic and privacy policies. We also interfaced between different NLP models of PoliCheck and Polisis to extract the purpose behind each data flow.

Our network-to-policy consistency analysis revealed that about 70% of data flows from VR apps were not disclosed or consistent with their privacy policies: only 30% were consistent. Furthermore, 38 apps did not have privacy policies, including apps from the official Oculus app store. Some app developers also had the tendency to neglect declaring data collected by the platform and third parties. We found that by automatically including these other parties' privacy policies in OVRSEEN's network-to-policy consistency analysis, 74% of data flows became consistent. We also found that 69% of data flows have purposes unrelated to the core function-

ality (*e.g.,* advertising, marketing campaigns, analytics), and only a handful of apps are explicit about notice and consent. OVRSEEN's implementation and datasets are made available at [62].

Overview. The rest of this paper is structured as follows. Section 2 provides background on the OVR platform and its data collection practices that motivate our work. Section 3 provides the methodology and results for OVRSEEN's network traffic analysis. Section 4 provides the methodology and results for OVRSEEN's policy analysis, network-to-policy consistency analysis, and purpose extraction. Section 5 discusses the findings and provides recommendations. Section 6 discusses related work. Section 7 concludes the paper.

2 Oculus VR Platform and Apps

In this paper, we focus on the Oculus VR (OVR), a representative of state-of-the art VR platform. A pioneer and leader in the VR space, OVR was bought by Facebook in 2014 [16] (we refer to both as "platform-party"), and it maintains to be the most popular VR platform today. Facebook has integrated more social features and analytics to OVR and now even requires users to sign in using a Facebook account [41].

We used the latest Oculus device, Quest 2, for testing. Quest 2 is completely wireless: it can operate standalone and run apps, without being connected to other devices. In contrast, *e.g.,* Sony Playstation VR needs to be connected to a Playstation 4 as its game controller. Quest 2 runs Oculus OS, a variant of Android 10 that has been modified and optimized to run VR environments and apps. The device comes with a few pre-installed apps, such as the Oculus browser. VR apps are usually developed using two popular game engines called Unity [65] and Unreal [15]. Unlike traditional Android apps that run on Android JVM, these 3D app development frameworks compile VR apps into optimized (*i.e.,* stripped) binaries to run on Quest 2 [66].

Oculus has an official app store and a number of third-party app stores. The Oculus app store offers a wide range of apps (many of them are paid), which are carefully curated and tested (*e.g.,* for VR motion sickness). In addition to the Oculus app store, we focus on SideQuest—the most popular third-party app store endorsed by Facebook [34]. In contrast to apps from the official store, apps available on SideQuest are typically at their early development stage and thus are mostly free. Many of them transition from SideQuest to the Oculus app store once they mature and become paid apps. As of March 2021, the official Oculus app store has 267 apps (79 free and 183 paid), and the SideQuest app store has 1,075 apps (859 free and 218 paid).

Motivation: privacy risks in OVR. VR introduces privacy risks, some of which are similar to other Internet-based platforms (*e.g.,* Android [12, 13], IoT devices [3, 17], Smart TVs [37, 67]), *etc.*), while others are unique to the VR platform. For example, VR headsets and hand controllers are equipped with sensors that collect data about the user's physical movement, body characteristics, voice activity, hand tracking, eye tracking, facial expressions, and play area [27,36,38], which may in turn reveal sensitive information about our physique, emotions, and home. Quest 2 can also act as a fitness tracker, thanks to the built-in Oculus Move app that tracks time spent for actively moving and amount of calories burned across all apps [43]. Furthermore, Oculus has been continuously updating their privacy policy with a trend of increasingly collecting more data over the years. Most notably, we observed a major update in May 2018, coinciding with the GDPR implementation date. Many apps have no privacy policy, or fail to properly include the privacy policies of third-party libraries. Please see Appendix A for more detail on observations that motivated our study, and Section 6 on related work. The privacy risks on the relatively new VR platform are not yet well understood.

Goal and approach: privacy analysis of OVR. In this paper, we seek to characterize the privacy risks introduced when potentially-sensitive data available on the device are sent by the VR apps and/or the platform to remote destinations for various purposes. We followed an experimental and data-driven approach, and we chose to test and analyze the most popular VR apps. In Section 3, we characterize the actual behavior exhibited in the network traffic generated by these VR apps and platform. In Section 4, we present how we downloaded the privacy policies of the selected VR apps, the platform, and relevant third-party libraries, used NLP to extract and analyze the statements made about data collection, analyzed their consistency when compared against the actual data flows found in traffic, and extracted the purpose of data collection.

App corpus. We selected OVR apps that are widely used by players. Our app corpus consists of 150 popular paid and free apps from both the official Oculus app store and SideQuest. In contrast, previous work typically considered only free apps from the official app store [12,13,37,67]. We used the number of ratings/reviews as the popularity metric, and considered only apps that received at least 3.5 stars. We selected three groups of 50 apps each: (1) the top-50 free apps and (2) the top-50 paid apps from the Oculus app store, and (3) the top-50 apps from the SideQuest store. We selected an equal number of paid and free apps from the Oculus app store to gain insight into both groups equally. We purposely did not just pick the top-100 apps, because paid apps tend to receive more reviews from users and this would bias our findings towards paid apps. Specifically, this would make our corpus consist of 90% paid and 10% free apps.

Our app corpus is representative of both app stores. Our top-50 free and top-50 paid Oculus apps constitute close to 40% of all apps on the Oculus app store, whereas the total number of downloads of our top-50 SideQuest apps is approximately

45% of all downloads for the SideQuest store. Out of these 150 apps, selected for their popularity and representativeness, we were able to decrypt and analyze the network traffic for 140 of them for reasons explained in Section 3.2.1.

3 OVRSEEN: Network Traffic

In this section, we detail our methodology for collecting and analyzing network traffic. In Section 3.1, we present OVRSEEN's system for collecting network traffic and highlight our decryption technique. Next, in Section 3.2, we describe our network traffic dataset and the extracted data flows. In Section 3.3, we report our findings on the OVR ATS ecosystem by identifying domains that were labeled as ATS by popular blocklists. Finally, in Section 3.4, we discuss data types exposures in the extracted data flows according to the context based on whether their destination is an ATS or not.

3.1 Network Traffic Collection

In this section, we present OVRSEEN's system for collecting and decrypting the network traffic that apps generate (① in Fig. 1). It is important to mention that OVRSEEN does not require rooting Quest 2, and as of June 2021, there are no known methods for doing so [23]. Since the Oculus OS is based on Android, we enhanced AntMonitor [54] to support the Oculus OS. Furthermore, to decrypt TLS traffic, we use Frida [45], a dynamic instrumentation toolkit. Using Frida to bypass certificate validation specifically for Quest 2 apps presents new technical challenges, compared to Android apps that have a different structure. Next, we describe these challenges and how we address them.

Traffic collection. For collecting network traffic, OVRSEEN integrates AntMonitor [54]—a VPN-based tool for Android that does not require root access. It runs completely on the device without the need to re-route traffic to a server. AntMonitor stores the collected traffic in PCAPNG format, where each packet is annotated (in the form of a PCAPNG comment) with the name of the corresponding app. To decrypt TLS connections, AntMonitor installs a user CA certificate. However, since Oculus OS is a modified version of Android 10, and AntMonitor only supports up to Android 7, we made multiple compatibility changes to support Oculus OS. In addition, we enhanced the way AntMonitor stores decrypted packets: we adjust the sequence and ack numbers to make packet re-assembly by common tools (*e.g.,* `tshark`) feasible in post-processing. We will submit a pull request to AntMonitor's open-source repository, so that other researchers can make use of it, not only on Quest 2, but also on other newer Android devices. For further details, see Appendix B.1.

TLS decryption. Newer Android devices, such as Quest 2, pose a challenge for TLS decryption: as of Android 7, apps that target API level 24 (Android 7.0) and above no longer trust user-added certificates [7]. Since Quest 2 cannot be rooted, we cannot install AntMonitor's certificate as a system certificate. Thus, to circumvent the mistrust of AntMonitor's certificate, OVRSEEN uses Frida (see Fig. 1) to intercept certificate validation APIs. To use Frida in a non-rooted environment, we extract each app and repackage it to include and start the Frida server when the app loads. The Frida server then listens to commands from a Frida client that is running on a PC using ADB. Although ADB typically requires a USB connection, we run ADB over TCP to be able to use Quest 2 wirelessly, allowing for free-roaming testing of VR apps.

OVRSEEN uses the Frida client to load and inject our custom JavaScript code that intercepts various APIs used to verify CA certificates. In general, Android and Quest 2 apps use three categories of libraries to validate certificates: (1) the standard Android library, (2) the Mbed TLS library [64] provided by the Unity SDK, and (3) the Unreal version of the OpenSSL library [14]. OVRSEEN places Frida hooks into the certificate validation functions provided by these three libraries. These hooks change the return value of the intercepted functions and set certain flags used to determine the validity of a certificate to ensure that AntMonitor's certificate is always trusted. While bypassing certificate validation in the standard Android library is a widely known technique [9], bypassing validation in Unity and Unreal SDKs is not. Thus, we developed the following technique.

Decrypting Unity and Unreal. Since most Quest 2 apps are developed using either the Unity or the Unreal game engines, they use the certificate validation functions provided by these engines instead of the ones in the standard Android library. Below, we present our implementation of certificate validation bypassing for each engine.

For Unity, we discovered that the main function that is responsible for checking the validity of certificates is `mbedtls_x509_crt_verify_with_profile()` in the Mbed TLS library, by inspecting its source code [6]. This library is used by the Unity framework as part of its SDK. Although Unity apps and its SDK are written in C#, the final Unity library is a C++ binary. When a Unity app is packaged for release, unused APIs and debugging symbols get removed from the Unity library's binary. This process makes it difficult to hook into Unity's functions since we cannot locate the address of a function of interest without having the symbol table to look up its address. Furthermore, since the binary also gets stripped of unused functions, we cannot rely on the debug versions of the binary to look up addresses because each app will have a different number of APIs included. To address this challenge, OVRSEEN automatically analyzes the debug versions of the non-stripped Unity binaries (provided by the Unity engine), extracts the function signature (*i.e.,* a set of hexadecimal numbers) of `mbedtls_x509_crt_verify_with_profile()`, and then looks for this signature in the stripped version of the binary

App Store	Apps	Domains	eSLDs	Packets	TCP Fl.
Oculus-Free	43	85	48	2,818	2,126
Oculus-Paid	49	54	35	2,278	1,883
SideQuest	48	57	40	2,679	2,260
Total			92		6,269

Table 1: **Network traffic dataset summary.** Note that the same domains and eSLDs can appear across the three groups of "App Store", so their totals are based on unique counts.

to find its address. This address can then be used to create the necessary Frida hook for an app. The details of this automated binary analysis can be found in Appendix B.2.

For Unreal, we discovered that the main function that is responsible for checking the validity of certificates is the function `x509_verify_cert()` in the OpenSSL library, which is integrated as part of the Unreal SDK. Fortunately, the Unreal SDK binary file comes with a partial symbol table that contains the location of `x509_verify_cert()`, and thus, OVRSEEN can set a Frida hook for it.

3.2 Network Traffic Dataset

3.2.1 Raw Network Traffic Data

We used OVRSEEN to collect network traffic for 140[1] apps in our corpus during the months of March and April 2021. To exercise these 140 apps and collect their traffic, we manually interacted with each one for seven minutes. Although there are existing tools that automate the exploration of regular (non-gaming) mobile apps (*e.g.,* [30]), automatic interaction with a variety of games is an open research problem. Fortunately, manual testing allows us to customize app exploration and split our testing time between exploring menus within the app to cover more of the potential behavior, and actually playing the game, which better captures the typical usage by a human user. As shown by prior work, such testing criteria lead to more diverse network traffic and reveal more privacy-related data flows [24, 50, 67]. Although our methodology might not be exhaustive, it is inline with prior work [37, 67].

Table 1 presents the summary of our network traffic dataset. We discovered 158 domains and 92 eSLDs in 6,269 TCP flows that contain 7,775 packets. Among the 140 apps, 96 were developed using the Unity framework, 31 were developed using the Unreal framework, and 13 were developed using other frameworks.

[1]The remaining 10 apps were excluded for the following reasons: (1) six apps could not be repackaged; (2) two apps were browser apps, which would open up the web ecosystem, diverting our focus from VR; (3) one app was no longer found on the store—we created our lists of top apps one month ahead of our experiments; and (4) one app could not start on the Quest 2 even without any of our modifications.

3.2.2 Network Data Flows Extracted

We processed the raw network traffic dataset and identified 1,135 data flows: ⟨*app, data type, destination*⟩. Next, we describe our methodology for extracting that information.

App names. For each network packet, the app name is obtained by AntMonitor [54]. This feature required a modification to work on Android 10, as described in Appendix B.1.

Data types. The data types we extracted from our network traffic dataset are listed in Table 3 and can be categorized into roughly three groups. First, we find personally identifiable information (*PII*), including: user identifiers (*e.g.,* Name, Email, and User ID), device identifiers (Android ID, Device ID, and Serial Number), Geolocation, *etc.* Second, we found system parameters and settings, whose combinations are known to be used by trackers to create unique profiles of users [37, 39], *i.e., Fingerprints*. Examples include various version information (*e.g.,* Build and SDK Versions), Flags (*e.g.,* indicating whether the device is rooted or not), Hardware Info (*e.g.,* Device Model, CPU Vendor, *etc.*), Usage Time, *etc.* Finally, we also find data types that are unique to VR devices (*e.g.,* VR Movement and VR Field of View) and group them as *VR Sensory Data*. These can be used to uniquely identify a user or convey sensitive information—the VR Play Area, for instance, can represent the actual area of the user's household.

We use several approaches to find these data types in HTTP headers and bodies, and also in any raw TCP segments that contain ASCII characters. First, we use string matching to search for data that is static by nature. For example, we search for user profile data (*e.g.,* User Name, Email, *etc.*) using our test OVR account and for any device identifiers (*e.g.,* Serial Number, Device ID, *etc.*) that can be retrieved by browsing the Quest 2 settings. In addition, we search for their MD5 and SHA1 hashes. Second, we utilize regular expressions to capture more dynamic data types. For example, we can capture different Unity SDK versions using `UnityPlayer/[\d.]+\d`. Finally, for cases where a packet contains structured data (*e.g.,* URL query parameters, HTTP Headers, JSON in HTTP body, *etc.*), we split the packet into key-value pairs and create a list of unique keys that appear in our entire network traffic dataset. We then examine this list to discover keys that can be used to further enhance our search for data types. For instance, we identified that the keys "user_id" and "x–playeruid" can be used to find User IDs. Appendix C.1 provides more details on our data types.

Destinations. To extract the destination fully qualified domain name (FQDN), we use the HTTP Host field and the TLS SNI (for cases where we could not decrypt the traffic). Using tldextract, we also identify the effective second-level domain (eSLD) and use it to determine the high level organization that owns it via Crunchbase. We also adopt similar labeling methodologies from [67] and [5] to categorize each destination as either *first-*, *platform-*, or *third-*party. To perform the

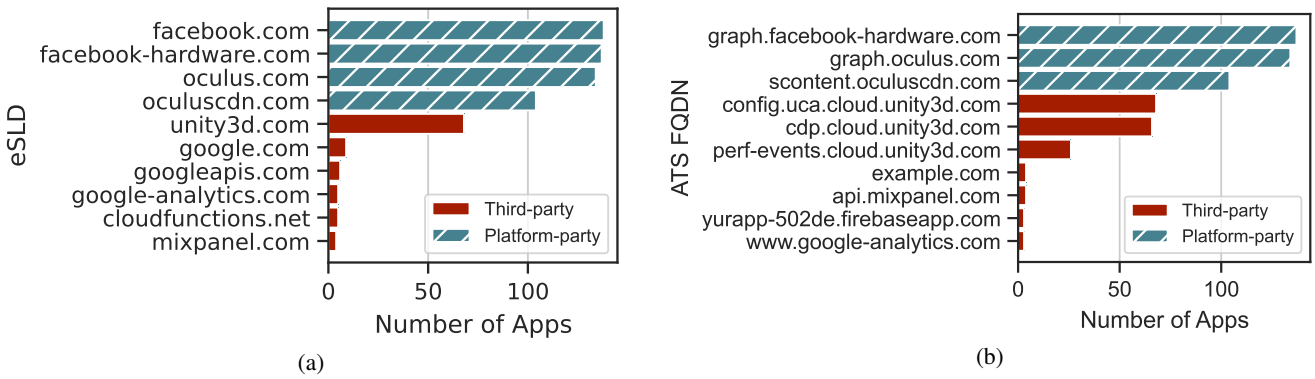


Figure 2: **Top-10 platform and third-party (a) eSLDs and (b) ATS FQDNs.** They are ordered by the number of apps that contact them. Each app may have a few first-party domains: we found that 46 out of 140 (33%) apps contact their own eSLDs.

categorization, we also make use of collected privacy policies (see Fig. 1 and Section 4), as described next. First, we tokenize the domain and the app's package name. We label a domain as first-party if the domain's tokens either appear in the app's privacy policy URL or match the package name's tokens. If the domain is part of cloud-based services (*e.g., vrapp.amazonaws.com*), we only consider the tokens in the subdomain (*vrapp*). Second, we categorize the destination as platform-party if the domain contains the keywords "oculus" or "facebook". Finally, we default to the third-party label. This means that the data collection is performed by an entity that is not associated with app developers nor the platform, and the developer may not have control of the data being collected. The next section presents further analysis of the destination domains.

3.3 OVR Advertising & Tracking Ecosystem

In this section, we explore the destination domains found in our network traffic dataset (see Section 3.2.2). Fig. 2a presents the top-10 eSLDs for platform and third-party. We found that, unlike the mobile ecosystem, the presence of third-parties is minimal and platform traffic dominates in all apps (*e.g., oculus.com*, *facebook.com*). The most prominent third-party organization is Unity (*e.g., unity3d.com*), which appears in 68 out of 140 apps (49%). This is expected since 96 apps in our dataset were developed using the Unity engine (see Section 3.2.1). Conversely, although 31 apps in our dataset were developed using the Unreal engine, it does not appear as a major third-party data collector because Unreal does not provide its own analytics service. Beyond Unity, other small players include Alphabet (*e.g., google.com*, *cloudfunctions.net*) and Amazon (*e.g., amazonaws.com*). In addition, 87 out of 140 apps contact four or fewer third-party eSLDs (62%).

Identifying ATS domains. To identify ATS domains, we apply the following popular domain-based blocklists: (1) *Pi-Hole's Default List* [46], a list that blocks cross-platform ATS domains for IoT devices; (2) *Mother of All Adblocking* [8], a list that blocks both ads and tracking domains for mobile devices; and (3) *Disconnect Me* [10], a list that blocks tracking domains. For the rest of the paper, we will refer to the above lists simply as "blocklists". We note that there are no blocklists that are curated for VR platforms. Thus, we choose blocklists that balance between IoT and mobile devices, and one that specializes in tracking.

OVR ATS ecosystem. The majority of identified ATS domains relate to social and analytics-based purposes. Fig. 2b provides the top-10 ATS FQDNs that are labeled by our blocklists. We found that the prevalent platform-related FQDNs along with Unity, the prominent third party, are labeled as ATS. This is expected: domains such as *graph.oculus.com* and *perf-events.cloud.unity3d.com* are utilized for social features like managing leaderboards and app analytics, respectively. We also consider the presence of organizations based on the number of unique domains contacted. The most popular organization is Alphabet, which has 13 domains, such as *google-analytics.com* and *firebase-settings.crashlytics.com*. Four domains are associated with Facebook, such as *graph.facebook.com*. Similarly, four are from Unity, such as *userreporting.cloud.unity3d.com* and *config.uca.cloud.unity3d.com*. Other domains are associated with analytics companies that focus on tracking how users interact with apps (*e.g.,* whether they sign up for an account) such as *logs-01.loggly.com*, *api.mixpanel.com*, and *api2.amplitude.com*. Lastly, we provide an in-depth comparison to other ecosystems in Section 5.1.

Missed by blocklists. The three blocklists that we use in OVRSEEN are not tailored for the Oculus platform. As a result, there could be domains that are ATS related but not labeled as such. To that end, we explored and leveraged data flows to find potential domains that are missed by blocklists. In particular, we start from data types exposed in our network traffic, and identify the destinations where these data types are sent to. Table 2 summarizes third-party destinations that collect the most data types and are *not* already captured by any of the blocklists. We found the presence of 11 different

FQDN	Organization	Data Types
bdb51.playfabapi.com	Microsoft	11
sharedprod.braincloudservers.com	bitHeads Inc.	8
cloud.liveswitch.io	Frozen Mountain Software	7
datarouter.ol.epicgames.com	Epic Games	6
9e0j15elj5.execute-api.us-west-1.amazonaws.com	Amazon	5

Table 2: Top-5 third-party FQDNs that are missed by blocklists based on the number of data types exposed.

Data Types (21)	Apps			FQDNs			% Blocked		
PII	1st	3rd	Pl.	1st	3rd	Pl.	1st	3rd	Pl.
Device ID	6	64	2	6	13	1	0	38	100
User ID	5	65	0	5	13	0	20	38	-
Android ID	6	31	18	6	7	2	17	43	50
Serial Number	0	0	18	0	0	2	-	-	50
Person Name	1	7	0	1	4	0	0	50	-
Email	2	5	0	2	5	0	0	20	-
Geolocation	0	5	0	0	4	0	-	50	-
Fingerprint									
SDK Version	23	69	20	34	28	4	6	46	0
Hardware Info	21	65	19	25	23	3	4	39	33
System Version	16	62	19	20	21	3	5	43	33
Session Info	7	66	2	7	13	1	14	46	100
App Name	4	65	2	4	10	1	25	40	100
Build Version	0	61	0	0	3	0	-	100	-
Flags	6	53	2	6	8	1	0	50	100
Usage Time	2	59	0	2	4	0	0	50	-
Language	5	28	16	5	9	1	0	56	0
Cookies	5	4	2	5	3	1	0	33	100
VR Sensory Data									
VR Play Area	0	40	0	0	1	0	-	100	-
VR Movement	1	24	2	1	6	1	0	67	100
VR Field of View	0	16	0	0	1	0	-	100	-
VR Pupillary Distance	0	16	0	0	1	0	-	100	-
Total	33	70	22	44	39	5	5	36	20

Table 3: **Data types exposed in the network traffic dataset.** Column "Apps" reports the number of apps that send the data type to a destination; column "FQDNs" reports the number of FQDNs that receive that data type; and column "% Blocked" reports the percentage of FQDNs blocked by blocklists. Using sub-columns, we denote party categories: first (1st), third (3rd), and platform (Pl.) parties.

organizations, not caught by blocklists, including: Microsoft, bitHeads Inc., and Epic Games—the company that created the Unreal engine. The majority are cloud-based services that provide social features, such as messaging, and the ability to track users for engagement and monetization (*e.g.,* promotions to different segments of users). We provide additional FQDNs missed by blocklists in Appendix C.2.

3.4 Data Flows in Context

The exposure of a particular data type, on its own, does not convey much information: it may be appropriate or inappropriate depending on the context [40]. For example, geolocation sent to the GoogleEarth VR or Wander VR app is necessary for the functionality, while geolocation used for ATS purposes is less appropriate. The network traffic can be used to partly infer the purpose of data flows, *e.g.,* depending on whether the destination being first-, third-, or platform-party; or an ATS. Table 3 lists all data types found in our network traffic, extracted using the methods explained in Section 3.2.2.

Third party. Half of the apps (70 out of 140) expose data flows to third-party FQDNs, 36% of which are labeled as ATS by blocklists. Third parties collect a number of PII data types, including Device ID (64 apps), User ID (65 apps), and Android ID (31 apps), indicating cross-app tracking. In addition, third parties collect system, hardware, and version info from over 60 apps—denoting the possibility that the data types are utilized to fingerprint users. Further, all VR specific data types, with the exception of VR Movement, are collected by a single third-party ATS domain belonging to Unity. VR Movement is collected by a diverse set of third-party destinations, such as *google-analytics.com*, *playfabapi.com* and *logs-01.loggly.com*, implying that trackers are becoming interested in collecting VR analytics.

Platform party. Our findings on exposures to platform-party domains are a lower bound since not all platform traffic could be decrypted (see Section 7). However, even with limited decryption, we see a number of exposures whose destinations are five third-party FQDNs. Although only one of these FQDNs is labeled as ATS by the blocklists, other platform-party FQDNs could be ATS domains that are missed by blocklists (see Section 3.3). For example, *graph.facebook.com* is an ATS FQDN, and *graph.oculus.com* appears to be its counterpart for OVR; it collects six different data types in our dataset. Notably, the platform party is the sole party responsible for collecting a sensitive hardware ID that cannot be reset by the user—the Serial Number. In contrast to OVR, the Android developer guide strongly discourages its use [19].

First party. Only 33 apps expose data flows to first-party FQDNs, and only 5% of them are labeled as ATS. Interestingly, the blocklists tend to have higher block rates for first-party FQDNs if they collect certain data types, *e.g.,* Android ID (17%), User ID (20%), and App Name (25%). Popular data types collected by first-party destinations are Hardware Info (21 apps), SDK Version (23 apps), and System Version (16 apps). For developers, this information can be used to prioritize bug fixes or improvements that would impact the most users. Thus, it makes sense that only ~5% of first-party FQDNs that collect this information are labeled as ATS.

Summary. The OVR ATS ecosystem is young when compared to Android and Smart TVs. It is dominated by tracking

domains for social features and analytics, but not by ads. We have detailed 21 different data types that OVR sends to first-, third-, and platform-parties. State-of-the-art blocklists only captured 36% of exposures to third parties, missing some sensitive exposures such as Email, User ID, and Device ID.

4 OVRSEEN: Privacy Policy Analysis

In this section, we turn our attention to the intended data collection and sharing practices, as stated in the text privacy policy. For example, from the text *"We may collect your email address and share it for advertising purposes"*, we want to extract the collection statement ("we", which implies the app's first-party entity; "collect" as action; and "email address" as data type) and the purpose ("advertising"). In Section 4.1.1, we present our methodology for extracting data collection statements, and comparing them against data flows found in network traffic for consistency. OVRSEEN builds and improves on state-of-the-art NLP-based tools: PoliCheck [5] and PolicyLint [4], previously developed for mobile apps. In Section 4.1.2, we present our VR-specific ontologies for data types and entities. In Section 4.1.3, we report network-to-policy consistency results. Section 4.2 describes how we interface between the different NLP models of PoliCheck and Polisis to extract the data collection purpose and other context for each data flow.

Collecting privacy policies. For each app in Section 3, we also collected its privacy policy on the same day that we collected its network traffic. Specifically, we used an automated Selenium [59] script to crawl the webstore and extracted URLs of privacy policies. For apps without a policy listed, we followed the link to the developer's website to find a privacy policy. We also included eight third-party policies (*e.g.,* from Unity, Google), referred to by the apps' policies.

For the top-50 free apps on the Oculus store, we found that only 34 out of the 43 apps have privacy policies. Surprisingly, for the top-50 paid apps, we found that only 39 out of 49 apps have privacy policies. For the top-50 apps on SideQuest, we found that only 29 out of 48 apps have privacy policies. Overall, among apps in our corpus, we found that only 102 (out of 140) apps provide valid English privacy policies. We treated the remaining apps as having empty privacy policies, ultimately leading OVRSEEN to classify their data flows as omitted disclosures.

4.1 Network-to-Policy Consistency

Our goal is to analyze text in the app's privacy policy, extract statements about data collection (and sharing), and compare them against the actual data flows found in network traffic.

4.1.1 Consistency Analysis System

OVRSEEN builds on state-of-the-art tools: PolicyLint [4] and PoliCheck [5]. PolicyLint [4] provides an NLP pipeline that takes a sentence as input. For example, it takes the sentence *"We may collect your email address and share it for advertising purposes"*, and extracts the collection statement "(entity: *we*, action: *collect*, data type: *email address*)". More generally, PolicyLint takes the app's privacy policy text, parses sentences and performs standard NLP processing, and eventually extracts data collection statements defined as the tuple $P = \langle$*app, data type, entity*$\rangle$, where *app* is the sender and *entity* is the recipient performing an *action* (collect or not collect) on the *data type*. PoliCheck [5] takes the app's data flows (extracted from the network traffic and defined as $F = \langle$*data type, entity*$\rangle$) and compares it against the stated P for consistency.

PoliCheck classifies the disclosure of F as *clear* (if the data flow exactly matches a collection statement), *vague* (if the data flow matches a collection statement in broader terms), *omitted* (if there is no collection statement corresponding to the data flow), *ambiguous* (if there are contradicting collection statements about a data flow), or *incorrect* (if there is a data flow for which the collection statement states otherwise). Following PoliCheck's terminology [5], we further group these five types of disclosures into two groups: *consistent* (clear and vague disclosures) and *inconsistent* (omitted, ambiguous, and incorrect) disclosures. The idea is that for consistent disclosures, there is a statement in the policy that matches the data type and entity, either clearly or vaguely. Table 4 provides real examples of data collection disclosures extracted from VR apps that we analyzed.

Consistency analysis relies on pre-built ontologies and synonym lists used to match (i) the data type and destination that appear in each F with (ii) any instance of P that discloses the same (or a broader) data type and destination[2]. OVRSEEN's adaptation of ontologies specifically for VR is described in Section 4.1.2. We also improved several aspects of PoliCheck, as described in detail in Appendix D.1. First, we added a feature to include a third-party privacy policy for analysis if it is mentioned in the app's policy. We found that 30% (31/102) of our apps' privacy policies reference third-party privacy policies, and the original PoliCheck would mislabel third-party data flows from these apps as omitted. Second, we added a feature to more accurately resolve first-party entity names. Previously, only first-person pronouns (*e.g.,* "we") were used to indicate a first-party reference, while some privacy policies use company and app names in first-party references. The original PoliCheck would incorrectly recognize these first-

[2]For example (see Fig. 3a), "email address" is a special case of "contact info" and, eventually, of "pii". There is a clear disclosure *w.r.t.* data type if the "email address" is found in a data flow and a collection statement. A vague disclosure is declared if the "email address" is found in a data flow and a collection statement that uses the term "pii" in the privacy policy. An omitted disclosure means that "email address" is found in a data flow, but there is no mention of it (or any of its broader terms) in the privacy policy.

	Disclosure Type	Privacy Policy Text	Action : Data Collection Statement (P)	Data Flow (F)
Consistent	Clear	"For example, we collect information ..., and *a timestamp for the request*."	*collect* : ⟨*com.cvr.terminus, usage time, we*⟩	⟨*usage time, we*⟩
	Vague	"We will share *your information (in some cases personal information)* with third-parties, ..."	*collect* : ⟨*com.HomeNetGames.WW1oculus, pii, third party*⟩	⟨*serial number, oculus*⟩ ⟨*android id, oculus*⟩
Inconsistent	Omitted	-	*collect* : ⟨*com.kluge.SynthRiders, -, -*⟩	⟨*system version, oculus*⟩ ⟨*sdk version, oculus*⟩ ⟨*hardware information, oculus*⟩
	Ambiguous	"..., Skydance *will not disclose any Personally Identifiable Information to third parties ...* your *Personally Identifiable Information will be disclosed to such third parties* and ..."	*collect* : ⟨*com.SDI.TWD, pii, third party*⟩	⟨*serial number, oculus*⟩ ⟨*android id, oculus*⟩
	Incorrect	"We *do not share our customer's personal information with unaffiliated third parties ...*"	*not_collect* : ⟨*com.downpourinteractive.onward, pii, third party*⟩	⟨*device id, unity*⟩ ⟨*user id, oculus*⟩

Table 4: **Examples to illustrate the types of disclosures identified by PoliCheck.** A *data collection statement (P)* is extracted from the privacy policy text and is defined as the tuple $P = \langle app, data\ type, entity \rangle$. *A data flow (F)* is extracted from the network traffic and is defined as $F = \langle data\ type, entity \rangle$. During the consistency analysis, each P can be mapped to zero, one, or more F.

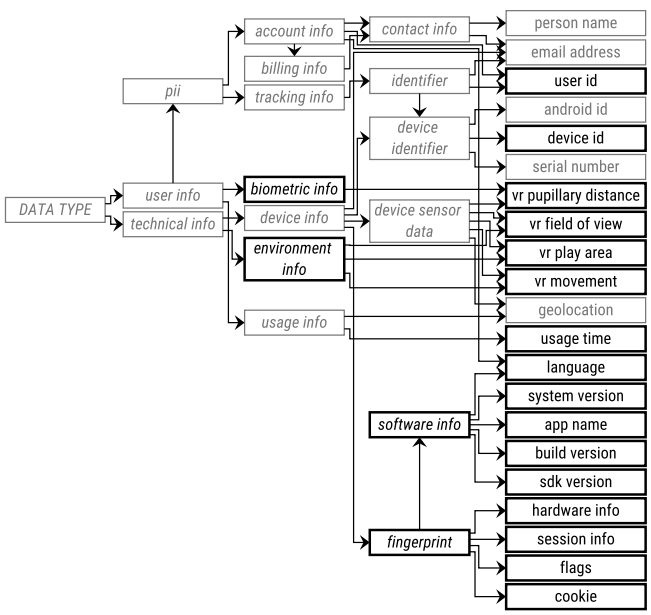
party references as third-party entities for 16% (16/102) of our apps' privacy policies.

4.1.2 Building Ontologies for VR

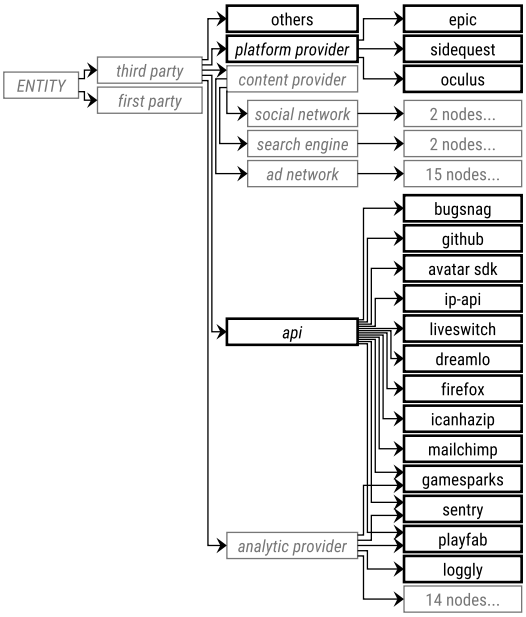
Ontologies are used to represent subsumptive relationships between terms: a link from term A to term B indicates that A is a broader term (*hypernym*) that subsumes B. There are two ontologies, namely data and entity ontologies: the data ontology maps data types and entity ontology maps destination entities. Since PoliCheck was originally designed for Android mobile app's privacy policies, it is important to adapt the ontologies to include data types and destinations specific to VR's privacy policies and actual data flows.

VR data ontology. Fig. 3a shows the data ontology we developed for VR apps. Leaf nodes correspond to all 21 data types found in the network traffic and listed in Table 3. Non-leaf nodes are broader terms extracted from privacy policies and may subsume more specific data types, *e.g.,* "device identifier" is a non-leaf node that subsumes "android id". We built a VR data ontology, starting from the original Android data ontology, in a few steps as follows. First, we cleaned up the original data ontology by removing data types that do not exist on OVR (*e.g.,* "IMEI", "SIM serial number", *etc.*). We also merged similar terms (*e.g.,* "account information" and "registration information") to make the structure clearer. Next, we used PoliCheck to parse privacy policies from VR apps. When PoliCheck parses the sentences in a privacy policy, it extracts terms and tries to match them with the nodes in the data ontology and the synonym list. If PoliCheck does not find a match for the term, it will save it in a log file. We inspected each term from this log file, and added it either as a new node in the data ontology or as a synonym to an existing term in the synonym list. Finally, we added new terms for data types identified in network traffic (see Section 3.4) as leaf nodes in the ontology. Most notably, we added VR-specific data types (see VR Sensory Data category shown in Table 3): "biometric info" and "environment info". The term "biometric info" includes physical characteristics of human body (*e.g.,* height, weight, voice, *etc.*); we found some VR apps that collect user's "pupillary distance" information. The term "environment information" includes VR-specific sensory information that describes the physical environment; we found some VR apps that collect user's "play area" and "movement". Table 5 shows the summary of the new VR data ontology. It consists of 63 nodes: 39 nodes are new in OVRSEEN's data ontology. Overall, the original Android data ontology was used to track 12 data types (*i.e.,* 12 leaf nodes) [5], whereas our VR data ontology is used to track 21 data types (*i.e.,* 21 leaf nodes) appearing in the network traffic (see Table 3 and Fig. 3a).

VR entity ontology. Entities are names of companies and other organizations which refer to destinations. We use a list of domain-to-entity mappings to determine which entity each domain belongs to (see Appendix D.1)—domain extraction and categorization as either first-, third-, or platform-party are described in detail in Section 3.2.2. We modified the Android entity ontology to adapt it to VR as follows: (1) we pruned entities that were not found in privacy policies of VR apps or in our network traffic dataset, and (2) we added new entities found in both sources. Table 5 summarizes the new entity ontology. It consists of 64 nodes: 21 nodes are new in OVRSEEN's entity ontology. Fig. 3b shows our VR entity ontology, in which we added two new non-leaf nodes: "platform provider" (which includes online distribution platforms or app stores that support the distribution of VR apps) and "api" (which refers to various third-party APIs and services that do not belong to existing entities). We identified 16 new entities that were not included in the original entity ontology. We visited the websites of those new entities and found that: three are platform providers, four are analytic providers, and 12 are service providers; these become the leaf nodes of "api". We also added a new leaf node called "others" to cover a few data flows, whose destinations cannot be determined from the IP address or domain name.



(a) Data Ontology



(b) Entity Ontology

Figure 3: **Ontologies for VR data flows.** Please recall that each data flow, F, is defined as $F = \langle data\ type,\ entity \rangle$. We started from the PoliCheck ontologies, originally developed for Android (printed in gray). First, we eliminated nodes that did not appear in our VR network traffic and privacy policies. Then, we added new leaf nodes (printed in black) based on new data types found in the VR network traffic and/or privacy policies text. Finally, we defined additional non-leaf nodes, such as "biometric info" and "api", in the resulting VR data and entity ontologies.

Platform	Data Ontology	Entity Ontology
Android [5]	38 nodes	209 nodes
OVR (OVRSEEN)	63 nodes	64 nodes
New nodes in OVR	39 nodes	21 nodes

Table 5: Comparison of PoliCheck and OVRSEEN Ontologies. Nodes include leaf nodes (21 data types and 16 entities) and non-leaf nodes (see Fig. 3).

Summary. Building VR ontologies has been non-trivial. We had to examine a list of more than 500 new terms and phrases that were not part of the original ontologies. Next, we had to decide whether to add a term into the ontology as a new node, or as a synonym to an existing node. In the meantime, we had to remove certain nodes irrelevant to VR and merge others because the original Android ontologies were partially machine-generated and not carefully curated.

4.1.3 Network-to-Policy Consistency Results

We ran OVRSEEN's privacy policy analyzer to perform network-to-policy consistency analysis. Please recall that we extracted 1,135 data flows from 140 apps (see Section 3.2.2).

OVR data flow consistency. In total, 68% (776/1,135) data flows are classified as inconsistent disclosures. The large majority of them 97% (752/776) are omitted disclosures, which are not declared at all in the apps' respective privacy policies.

Fig. 4 presents the data-flow-to-policy consistency analysis results. Out of 93 apps which expose data types, 82 apps have at least one inconsistent data flows. Among the remaining 32% (359/1,135) consistent data flows, 86% (309/359) are classified as vague disclosures. They are declared in vague terms in the privacy policies (*e.g.,* the app's data flows contain the data type "email address", whereas its privacy policy only declares that the app collects "personal information"). Clear disclosures are found in only 16 apps.

Data type consistency. Fig. 5a reports network-to-policy consistency analysis results by data types—recall that in Section 3.2.2 we introduced all the exposed data types into three categories: *PII*, *Fingerprint*, and *VR Sensory Data*. The PII category contributes to 22% (250/1,135) of all data flows. Among the three categories, PII has the best consistency: 57% (142/250) data flows in this category are classified as consistent disclosures. These data types are well understood and also treated as PII in other platforms. On Android [5], it is reported that 59% of PII flows were consistent—this is similar to our observation on OVR. The Fingerprint category constitutes 69% (784/1,135) of all data flows: around 25% (199/784) of data flows in this category are classified as consistent disclosures. The VR Sensory Data category constitutes around 9% (101/1,135) of all data flows—this category is unique to the VR platform. Only 18% (18/101) data flows of this category are consistent—this indicates that the collection of data types in this category is not properly disclosed in privacy policies.

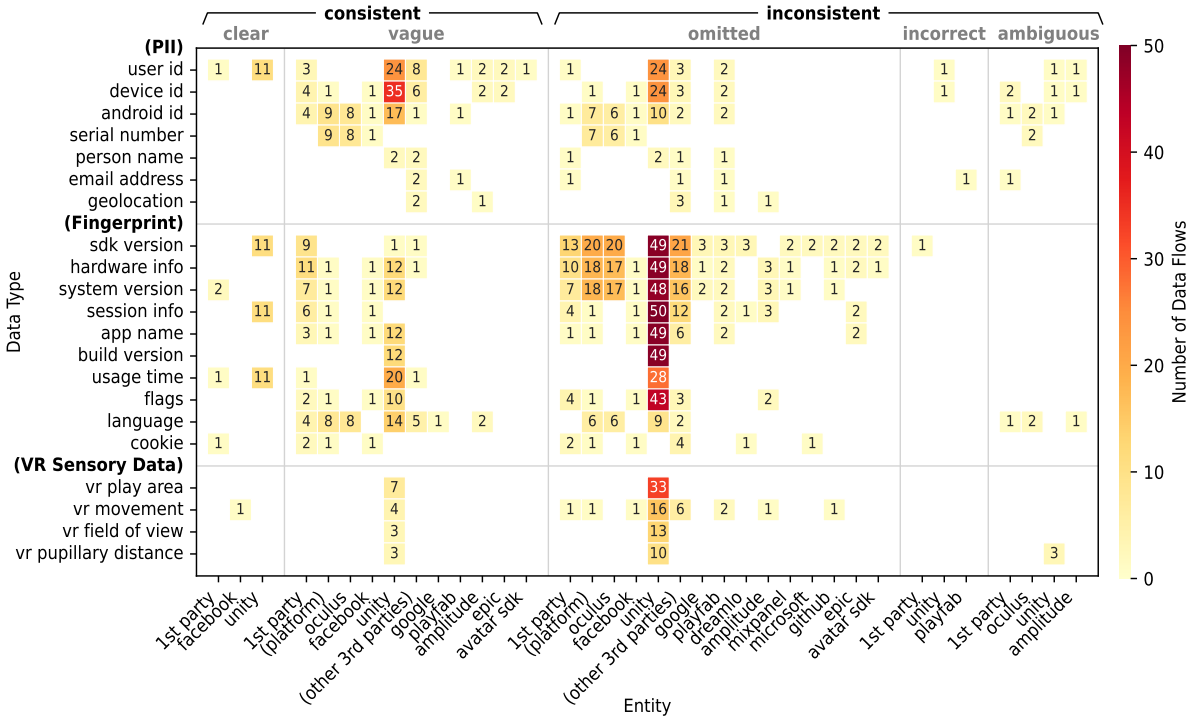


Figure 4: **Summary of network-to-policy consistency analysis results.** Columns whose labels are in parentheses provide aggregate values: *e.g.,* column "(platform)" aggregates the columns "oculus" and "facebook"; column "(other 3rd parties)" aggregates the subsequent columns. The numbers count data flows; each data flow is defined as ⟨*app, data type, destination*⟩).

Entity consistency. Fig. 5b reports our network-to-policy consistency results, by entities. Only 29% (298/1,022) of third-party and platform data flows are classified as consistent disclosures. First-party data flows constitute 10% (113/1,135) of all data flows: 54% (61/113) of these first-party data flows are classified as consistent disclosures. Thus, 69% (785/1,135) of all data flows are classified as inconsistent disclosures. Third-party and platform data flows constitute 90% (1,022/1,135) of all data flows—surprisingly, only 29% (298/1,022) of these third-party and platform data flows are classified as consistent disclosures.

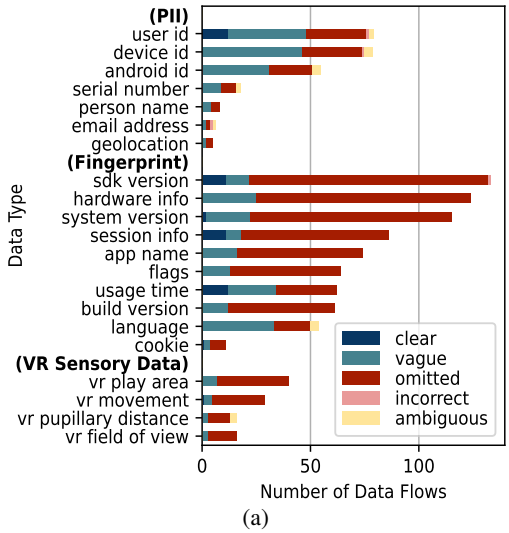
Unity is the most popular third-party entity, with 66% (746/1,135) of all data flows. Only 31% (232/746) of these Unity data flows are classified as consistent, while the majority (69%) are classified as inconsistent disclosures. Platform (*i.e.,* Oculus and Facebook) data flows account for 11% (122/1,135) of all data flows; only 28% (34/122) of them are classified as consistent disclosures. Other less prevalent entities account only around 14% (154/1,135) of all data flows.

Referencing Oculus and Unity privacy policies. Privacy policies can link to each other. For instance, when using Quest 2, users should be expected to consent to the Oculus privacy policy (for OVR). Likewise, when app developers utilize a third party engine (*e.g.,* Unity) their privacy policies should include the Unity privacy policy. To the best of our knowledge, this aspect has not been considered in prior work [5, 29, 72].

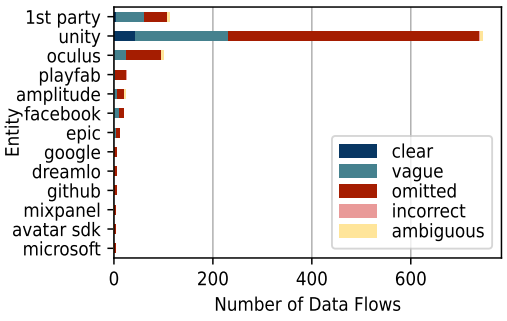
Interestingly, when we included the Oculus and Unity privacy policies (when applicable) in addition to the app's own privacy policy, we found that the majority of platform (116/122 or 96%) and Unity (725/746 or 97%) data flows get classified as consistent disclosures. Thus, 74% (841/1,135) of all data flows get classified as consistent disclosures. Fig. 6 shows the comparison of the results from this new experiment with the previous results shown in Fig. 5b. These show that data flows are properly disclosed in Unity and Oculus privacy policies even though the app developers' privacy policies usually do not refer to these two important privacy policies. Furthermore, we noticed that the Oculus and Unity privacy policies are well-written and clearly disclose collected data types. As discussed in [5], developers may be unaware of their responsibility to disclose third-party data collections, or they may not know exactly how third-party SDKs in their apps collect data from users. This is a recommendation for future improvement.

Validation of PoliCheck results (network-to-policy consistency). To test the correctness of PoliCheck when applied to VR apps, we manually inspected all data flows from apps that provided a privacy policy, and checked their consistency with corresponding collection statements in the policy. Three authors had to agree on the consistency result (one of the five disclosure types) of each data flow. We found the following.

First, we considered *multi-class classification* into consistent, omitted and incorrect disclosures, similar to PoliCheck's evaluation [5]. The performance of multi-class classification



(a)



(b)

Figure 5: Network-to-policy consistency analysis results aggregated by (a) data types, and (b) destination entities.

can be assessed using micro-averaging or macro-averaging of metrics across classes. *Micro-averaging* is more appropriate for imbalanced datasets and was also used for consistency analysis of Android apps [5] and Alexa skills [29]. In our VR dataset, we obtained 84% micro-averaged precision, recall and F1-score[3]. This is comparable to the corresponding numbers when applying PoliCheck to mobile [5] and Alexa Skills [29], which reported 90.8% and 83.3% (micro-averaged) precision/recall/F1-score, respectively. For completeness, we also computed the *macro-averaged* precision, recall and F1-score to be 74%, 89%, and 81% respectively (see Table 8).

Second, we considered the binary classification case (*i.e.,* we treat inconsistent disclosures as positive and consistent disclosures as negative samples). We obtained 77% precision, 94% recall, and 85% F1-score (see Appendix D.2 for more details). Overall, PoliCheck, along with our improvements for OVRSEEN, works well on VR apps[4].

[3]In multi-class classification, every misclassification is a false positive for one class and a false negative for other classes; thus, micro-averaged precision, recall, and F1-score are all the same (see Appendix D.2).

[4]However, the precision is lower when distinguishing between clear and vague disclosures. Our validation shows 23% vague disclosures were actually clearly disclosed. This is because OVRSEEN's privacy policy analyzer inherits the limitations of PoliCheck's NLP model which cannot extract data types and entities from a collection statement that spans multiple sentences..

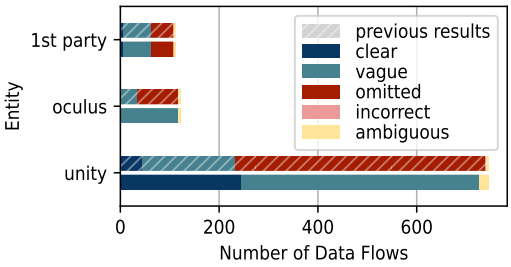


Figure 6: **Referencing Oculus and Unity privacy policies.** Comparing the results from the ideal case (including Unity and Oculus privacy policies by default) and the previous actual results (only including the app's privacy policy and any third-party privacy policies linked explicitly therein).

4.2 Data Collection in Context

Consistent (*i.e.,* clear, or even vague) disclosures are desirable because they notify the user about the VR apps' data collection and sharing practices. However, they are not sufficient to determine whether the information flow is within its context or social norms. This context includes (but is not limited to) the purpose and use, notice and consent, whether it is legally required, and other aspects of the "transmission principle" in the terminology of contextual integrity [40]. In the previous section, we have discussed the *consistency* of the network traffic *w.r.t.* the privacy policy statements: this provides some context. In this section, we identify an additional context: we focus on the *purpose* of data collection.

Purpose. We extract purpose from the app's privacy policy using Polisis [21]—an online privacy policy analysis service based on deep learning. Polisis annotates privacy policy texts with purposes at text-segment level. We developed a translation layer to map annotated purposes from Polisis into consistent data flows (see Appendix D.3). This mapping is possible only for data flows with consistent disclosures, since we need the policy to extract the purpose of a data flow. We were able to process 293 (out of 359) consistent data flows[5] that correspond to 141 text segments annotated by Polisis. Out of the 293 data flows, 69 correspond to text segments annotated as "unspecific", *i.e.,* Polisis extracted no purpose. The remaining 224 data flows correspond to text segments annotated with purposes. Since a data flow can be associated with multiple purposes, we expanded the 224 into 370 data flows, so that each data flow has exactly one purpose. There are nine distinct purposes identified by Polisis (including advertising, analytics, personalization, legal, *etc.*; see Fig. 7).

[5]Polisis did not process the text segments that correspond to the remaining 66 consistent data flows: it did not annotate the text segments and simply reported that their texts were too short to analyze.

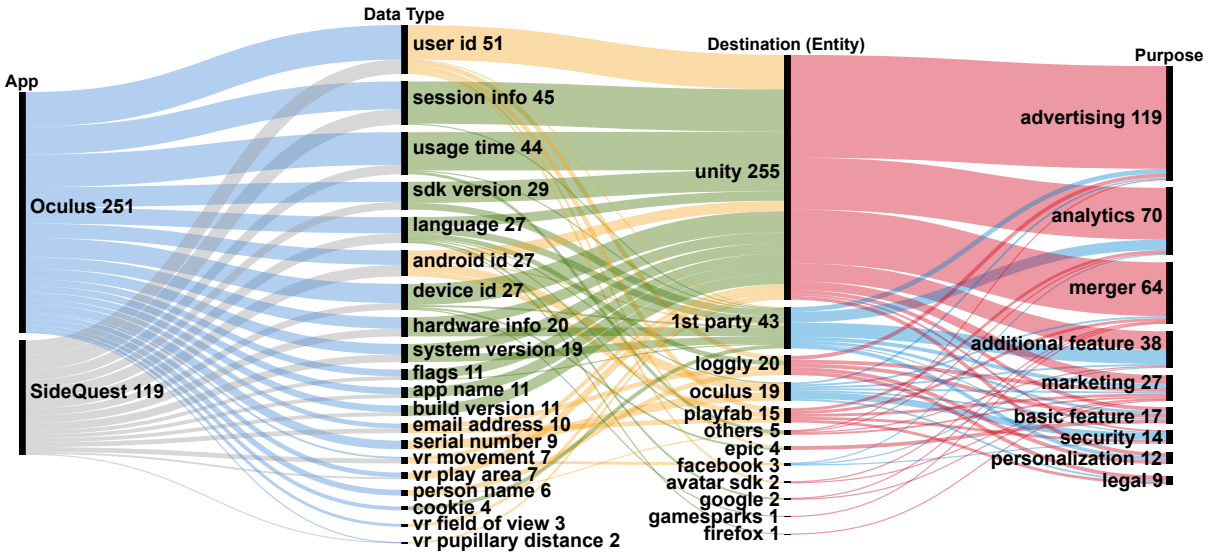


Figure 7: **Data flows in context.** We consider the data flows (⟨*app, data type, destination*⟩) found in the network traffic, and, in particular, the 370 data flows associated with consistent disclosures. We analyze these in conjunction with their *purpose* as extracted from the privacy policy text and depict the augmented tuples ⟨*app, data type, destination, purpose* ⟩ in the above alluvial diagram. The diagram is read from left to right, for example: (1) out of 251 data flows from the Oculus app store, no more than 51 data flows collect User ID and send it to various destinations; (2) the majority of User ID is collected by Unity; and (3) Unity is responsible for the majority of data flows with the purpose of advertising. Finally, the color scheme of the edges helps keep track of the flow. From App to Data Type, the color indicates the app store: blue for Oculus apps and gray for SideQuest apps. From Data Type to Destination, the color indicates the type of data collected: PII and VR Sensory Data data flows are in orange, while Fingerprinting data flows are in green. From Destination to Purpose, we use blue to denote first-party destinations and red to denote third-party destinations.

To further understand whether data collection is essential to app functionality, we distinguish between purposes that support core functionality (*i.e.,* basic features, security, personalization, legal purposes, and merger) and those unrelated to core functionality (*i.e.,* advertising, analytics, marketing, and additional features) [35]. Intuitively, core functionality indicates services that users expect from an app, such as reading articles from a news app or making a purchase with a shopping app. We found that only 31% (116/370) of all data flows are related to core functionality, while 69% (254/370) are unrelated. Interestingly, 81% (94/116) of core-functionality-related data flows are associated with third-party entities, indicating that app developers use cloud services. On the other hand, data collection purposes unrelated to core functionality can be used for marketing emails or cross-platform targeted advertisements. This is partly also corroborated by our ATS findings in Section 3.3: 83% (211/254) are associated with third-party tracking entities. In OVR, data types can be collected for tracking purposes and used for ads on other mediums (*e.g.,* Facebook website) and not on the Quest 2 device itself.

Next, we looked into the data types exposed for different purposes. The majority of data flows related to core functionality (56% or 65/116) expose PII data types, while Fingerprinting data types appear in most (66% or 173/254) data flows unrelated to functionality. We found that 15 data types are collected for functionality: these are comprised of Fingerprinting (41% or 48/116 data flows) and VR Sensory Data (3% or 3/116 data flows). We found that 19 data types are collected for purposes unrelated to functionality: these are comprised of PII (26% or 65/254 data flows) and VR Sensory Data (6% or 16/254 data flows). Interestingly, VR Movement, VR Play Area, and VR Field of View are mainly used for "advertising", while VR Movement and VR Pupillary Distance are used for "basic features", "security", and "merger" purposes [21].

Validation of Polisis results (purpose extraction). In order to validate the results pertaining to purpose extraction, we read all the 141 text segments previously annotated by Polisis. Then, we manually annotated each text segment with one or more purposes (based on the nine distinct purposes identified by Polisis). We had three authors look at each segment independently and then agree upon its annotation. We then compared our annotation with the purpose output by Polisis for the same segment. We found that this purpose extraction has 80%, 79%, and 78% micro-averaged precision, recall, and F1-score respectively[6]. These micro-averaged results are directly comparable to the Polisis' results in [21]: OVRSEEN's purpose extraction works well on VR apps. For completeness, we also computed the macro-averaged precision, recall, and

[6]Please note that this is multi-label classification. Thus, unlike multi-class classification for PoliCheck, precision, recall, and F1-score are different.

F1-score: 81%, 78%, and 78%, respectively. Table 9 in Appendix D.3 reports the precision, recall, and F1-score for each purpose classification, and their micro- and macro-averages.

5 Discussion

5.1 VR-Specific Considerations

VR tracking has unique aspects and trends compared to other ecosystems, including but not limited to the following.

VR ads. The VR advertising ecosystem is currently at its infancy. Our analysis of destinations from the network traffic revealed that ad-related activity was missing entirely from OVR at the time of our experiments. Facebook recently started testing on-device ads for Oculus in June 2021 [44]. Ads on VR platforms will be immersive experiences instead of flat visual images; for example, Unity uses separate virtual rooms for ads [63]. We expect that tracking will further expand once advertising comes into VR (*e.g.,* to include tracking how users interact and behave within the virtual ad space). As VR advertising and tracking evolve, our OVRSEEN methodology, system, and datasets will continue to enable analysis that was not previously possible on any VR platforms.

Comparison to other ecosystems. Our analysis showed that the major players in the OVR tracking ecosystem are currently Facebook and Unity (see Fig. 2 and 5). The more established ecosystems such as mobile and Smart TVs are dominated by Alphabet [25,67]; they also have a more diverse playing field of trackers (*e.g.,* Amazon, Comscore Inc., and Adobe)—spanning hundreds of tracking destinations [25, 55, 67]. OVR currently has only a few players (*e.g.,* Facebook, Unity, Epic, and Alphabet). OVRSEEN can be a useful tool for continuing the study on this developing ecosystem.

Sensitive data. Compared to other devices, such as mobile, Smart TVs and IoT, the type of data that can be collected from a VR headset is arguably more sensitive. For example, OVR has access to various biometric information (*e.g.,* pupillary distance, hand geometry, and body motion tracking data) that can be used to identify users and even infer their health [43]. A study by Miller *et al.* [36] revealed the feasibility of identifying users with a simple machine learning model using less than five minutes of body motion tracking data from a VR device. Our experiments found evidence of apps collecting data types that are unique to VR, including biometric-related data types (see Section 3.2.2). While the numbers we found are small so far, with the developing VR tracking ecosystem, it is important to have a system such as OVRSEEN to detect the increasing collection of sensitive data over time.

Generalization. Within OVR, we only used OVRSEEN to analyze 140 apps in our corpus. However, we believe that it can be applied to other OVR apps, as long as they are created according to OVR standards. Beyond OVR, the network traffic analysis and network-to-policy consistency analysis can also be applied to other platforms, as long as their network traffic can be decrypted, as was the case with prior work on Android, Smart TV, *etc.* [37, 48, 54, 67].

5.2 Recommendations

Based on our findings, we provide recommendations for the OVR platform and developers to improve their data transparency practices.

Provide a privacy policy. We found that 38 out of the 140 popular apps, out of which 19 are from the Oculus app store, did not provide any privacy policy at all. Furthermore, 97% of inconsistent data flow disclosures were due to omitted disclosures by these 38 apps missing privacy policies (see Section 4). We recommend that the OVR platform require developers to provide a privacy policy for their apps, especially those available on the official Oculus app store.

Reference other parties' privacy policies. Developers are not the only ones collecting data during the usage of an app: third-parties (*e.g.,* Unity, Microsoft) and platform-party (*e.g.,* Oculus/Facebook) can also collect data. We found that 81 out of 102 app privacy policies did not reference policies of third-party libraries used by the app. We recommend that developers reference third-party and platform-party privacy policies. If they do that, then the consistency of disclosures will be quite high: up to 74% of data flows in the network traffic we collected (see Section 4.1.3). This indicates that, at least at this early stage, the VR ecosystem is better behaved than the mobile tracking ecosystem.

Notice and consent. We found that fewer than 10 out of 102 apps that provide a privacy policy explicitly ask users to read it and give consent to data collection (*e.g.,* for analytics purposes) upon first opening the app. We recommend that developers provide notice and ask for users' consent (*e.g.,* when a user launches the app for the first time) for data collection and sharing, as required by privacy laws such as GDPR [70].

Notifying developers. We contacted Oculus as well as the developers of the 140 apps that we tested. We provided courtesy notifications of the specific data flows and consistency we identified in their apps, along with recommendations. We received 24 responses (see the details in Appendix E). Developers were, in general, appreciative of the information and willing to adopt recommendations to improve their privacy policies. Several indicated they did not have the training or tools to ensure consistent disclosures.

6 Related Work

Privacy in Context. The framework of "Privacy in Context" [40] specifies the following aspects of information flow:

(1) actors: sender, recipient, subject; (2) type of information; and (3) transmission principle. The goal is to determine whether the information flow is appropriate within its context. The "transmission principle" is key in determining the appropriateness of the flow and may include: the purpose of data collection, notice and consent, required by law, *etc.* [40]. In this paper, we seek to provide context for the data flows (⟨*app, data type, destination*⟩) found in the network traffic. We primarily focus on the network-to-policy *consistency*, *purpose* of data collection, and we briefly comment on notice and consent. Most prior work on network analysis only characterized destinations (first *vs.* third parties, ATS, *etc.*) or data types exposed without additional contexts. One exception is MobiPurpose [24], which inferred data collection purposes of mobile (not VR) apps, using network traffic and app features (*e.g.,* URL paths, app metadata, domain name, *etc.*); the authors stated that "the purpose interpretation can be subjective and ambiguous". Our notion of purpose is explicitly stated in the privacy policies and/or indicated by the destination domain matching ATS blocklists. Shvartzshnaider *et al.* introduced the contextual integrity (CI) framework to understand and evaluate privacy policies [57]—they, however, leveraged manual inspection and not automation.

Privacy of various platforms. The research community has looked into privacy risks in various platforms, using static or dynamic code analysis, and—most relevant to us—network traffic analysis. Enck *et al.* performed static analysis of Android apps [13] and discovered PII misuse (*e.g.,* personal/phone identifiers) and ATS activity. Taintdroid, first introduced taint tracking for mobile apps [12]. Ren *et al.* [49] did a comprehensive evaluation of information exposure on smart home IoT devices. Moghaddam *et al.* and Varmarken *et al.* observed the prevalence of PII exposures and ATS activity [37, 67] in Smart TVs. Lentzsch *et al.* [29] performed a comprehensive evaluation on Alexa, a voice assistant platform. Ren *et al.* [50], Razaghpanah *et al.* [48], and Shuba *et al.* [54–56] developed tools for analysis of network traffic generated by mobile apps, and inspection for privacy exposures and ATS activity. Our work is the first to perform network traffic analysis on the emerging OVR platform, using dynamic analysis to capture and decrypt networking traffic on the device; this is more challenging for Unity and Unreal based apps because, unlike prior work that dealt with standard Android APIs, we had to deal with stripped binary files (*i.e.,* no symbol table). Augmented reality (AR) is another platform the research community has been focusing on in the past decade [1, 26, 28, 47, 51, 69]. While AR augments our perception and interaction with the real world, VR replaces the real world with a virtual one. Nevertheless, some AR privacy issues are similar to those in VR since they have similar sensors, *e.g.,* motion sensors.

Privacy of VR. Although there is work on security aspects of VR devices (*e.g.,* authentication and attacks on using virtual keyboards) [11, 31–33], the privacy of VR is currently not fully understood. Adams *et al.* [2] interviewed VR users and developers on security and privacy concerns, and learnt that they were concerned with data collection potentially performed by VR devices (*e.g.,* sensors, device being always on) and that they did not trust VR manufacturers (*e.g.,* Facebook owning Oculus). Miller *et al.* present a study on the implications of the ability of VR technology to track body motions [36]. Our work is motivated by these concerns but goes beyond user surveys to analyze data collection practices exhibited in the network traffic and stated in privacy policies.

Privacy policy analysis. Privacy policy and consistency analysis in various app ecosystems [4, 5, 21, 58, 68, 71, 72] is becoming increasingly automated. Privee [71] is a privacy policy analyzer that uses NLP to classify the content of a website privacy policy using a set of binary questions. Slavin *et al.* used static code analysis, ontologies, and information flow analysis to analyze privacy policies for mobile apps on Android [58]. Wang *et al.* applied similar techniques to check for privacy leaks from user-entered data in GUI [68]. Zimmeck *et al.* also leveraged static code analysis for privacy policy consistency analysis [72]; they improved on previous work by attempting to comply with legal requirements (*e.g.,* first *vs.* third party, negative policy statements, *etc.*). In Section 4, we leverage two state-of-the-art tools, namely PoliCheck [5] and Polisis [21], to perform data-flow-to-policy consistency analysis and to extract the data collection purpose, respectively. PoliCheck was built on top of PolicyLint [4], a privacy policy analyzer for mobile apps. It analyzes both positive and negative data collection (and sharing) statements, and detects contradictions. Lentzsch *et al.* also used off-the-shelf PoliCheck using a data ontology crafted for Alexa skills. OVRSEEN focuses on OVR and improves on PoliCheck in several ways, including VR-specific ontologies, referencing third-party policies, and extracting data collection purposes.

7 Conclusion

Summary. We present the first comprehensive study of privacy aspects for Oculus VR (OVR), the most popular VR platform. We developed OVRSEEN, a methodology and system to characterize the data collection and sharing practices of the OVR ecosystem by (1) capturing and analyzing data flows found in the network traffic of 140 popular OVR apps, and (2) providing additional contexts via privacy policy analysis that checks for consistency and identifies the purpose of data collection. We make OVRSEEN's implementation and datasets publicly available at [62]. This is the extended version of our paper, with the same title, published at USENIX Security Symposium 2022. Please take a look at our project page for more information [62].

Limitations and future directions. On the networking side, we were able to decrypt, for the first time, traffic of OVR

apps, but the OVR platform itself is closed and we could not decrypt most of its traffic. In future work, we will explore the possibility of addressing this limitation by further exploring binary analysis. On the privacy policy side, PoliCheck and Polisis rely on different underlying NLP model, with inherent limitations and incompatibilities—this motivates future work on a unified privacy policy and context analyzer.

Acknowledgment

This project was supported by NSF Awards 1815666 and 1956393. We would like to thank our shepherd, Tara Whalen, and the USENIX Security 2022 reviewers for their feedback, which helped to significantly improve the paper. We would also like to thank Yiyu Qian, for his help with part of our data collection process.

References

[1] A. Acquisti, R. Gross, and F. D. Stutzman. Face Recognition and Privacy in the Age of Augmented Reality. *Journal of Privacy and Confidentiality*, 6(2):1, 2014.

[2] D. Adams, A. Bah, C. Barwulor, N. Musaby, K. Pitkin, and E. M. Redmiles. Ethics Emerging: the Story of Privacy and Security Perceptions in Virtual Reality. In *SOUPS*, Aug. 2018.

[3] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. SoK: Security Evaluation of Home-Based IoT Deployments. In *IEEE SP*, 2019.

[4] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie. PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play. In *USENIX Security*, Aug. 2019.

[5] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman. Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck. In *USENIX Security*, Aug. 2020.

[6] ARMmbed. mbedtls: x509_crt.c. https://github.com/ARMmbed/mbedtls/blob/development/library/x509_crt.c, 2021.

[7] C. Brubaker and Android Security team. Changes to trusted certificate authorities in android nougat. https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html, July 2016.

[8] BSDgeek_Jake (XDA Developer). Moaab: Mother of all ad-blocking. https://forum.xda-developers.com/showthread.php?t=1916098, 2019.

[9] P. Cipolloni. Universal android ssl pinning bypass with frida. https://techblog.mediaservice.net/2017/07/universal-android-ssl-pinning-bypass-with-frida/, July 2017.

[10] Disconnect, Inc. disconnect-tracking-protection: Canonical repository for the disconnect services file. https://github.com/disconnectme/disconnect-tracking-protection, 2021.

[11] R. Duezguen, P. Mayer, S. Das, and M. Volkamer. Towards Secure and Usable Authentication for Augmented and Virtual Reality Head-Mounted Displays. *arXiv preprint arXiv:2007.11663*, 2020.

[12] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *OSDI*, Oct. 2010.

[13] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *USENIX Security*, Aug. 2011.

[14] Epic Games, Inc. Openssl (unreal version). https://github.com/EpicGames/UnrealEngine/tree/master/Engine/Source/ThirdParty/OpenSSL, 2021.

[15] Epic Games, Inc. Unreal engine. https://www.unrealengine.com/, 2021.

[16] Facebook. Facebook to acquire oculus. https://about.fb.com/news/2014/03/facebook-to-acquire-oculus/, March 2014.

[17] E. Fernandes, J. Jung, and A. Prakash. Security Analysis of Emerging Smart Home Applications. In *IEEE SP*, 2016.

[18] Google. Android developers - behavior changes: all apps. https://developer.android.com/about/versions/10/behavior-changes-all, 2021.

[19] Google. Android developers - best practices for unique. https://developer.android.com/training/articles/user-data-ids, 2021.

[20] Google. Android developers - privacy changes in android 10. https://developer.android.com/about/versions/10/privacy/changes, 2021.

[21] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer. Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning. In *USENIX Security*, Aug. 2018.

[22] S. Hayden. Oculus quest 2 surpasses original quest in monthly active users. https://www.roadtovr.com/oculus-quest-2-monthly-active-users/, January 2021.

[23] D. Heaney. The oculus quest 2 'jailbreak' seems to be fake. https://uploadvr.com/oculus-quest-2-jailbreak-seems-fake/, November 2020.

[24] H. Jin, M. Liu, K. Dodhia, Y. Li, G. Srivastava, M. Fredrikson, Y. Agarwal, and J. I. Hong. Why Are They Collecting My Data? Inferring the Purposes of Network Traffic in Mobile Apps. In *ACM IMWUT*, 2018.

[25] K. Kollnig, A. Shuba, R. Binns, M. V. Kleek, and N. Shadbolt. Are iPhones Really Better for Privacy? Comparative Study of iOS and Android Apps. *arXiv preprint arXiv:2109.13722*, 2021.

[26] A. Kotsios. Privacy in an Augmented Reality. *International Journal of Law and Information Technology*, 23(2):157–185, 2015.

[27] B. Lang. Where to change quest 2 privacy settings and see your vr data collected by facebook. https://www.roadtovr.com/oculus-quest-2-privacy-facebook-data-collection-settings/, October 2020.

[28] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. Towards Security and Privacy for Multi-User Augmented Reality: Foundations with End Users. In *IEEE SP*, 2018.

[29] C. Lentzsch, S. J. Shah, B. Andow, M. Degeling, A. Das, and W. Enck. Hey Alexa, is this Skill Safe?: Taking a Closer Look at the Alexa Skill Ecosystem. In *NDSS*, 2021.

[30] Y. Li, Z. Yang, Y. Guo, and X. Chen. DroidBot: a Lightweight UI-Guided Test Input Generator for Android. In *IEEE/ACM ICSE-C*, 2017.

[31] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, and X. Fu. I Know What You Enter on Gear VR. In *IEEE CNS*, 2019.

[32] S. Luo, A. Nguyen, C. Song, F. Lin, W. Xu, and Z. Yan. OcuLock: Exploring Human Visual System for Authentication in Virtual Reality Head-mounted Display. In *NDSS*, 2020.

[33] F. Mathis, J. H. Williamson, K. Vaniea, and M. Khamis. Fast and Secure Authentication in Virtual Reality using Coordinated 3D Manipulation and Pointing. *ACM ToCHI*, 2021.

[34] L. Matney. The oculus quest's unofficial app store gets backing from oculus founder palmer luckey. https://techcrunch.com/2020/09/23/the-oculus-quests-unofficial-app-store-gets-backing-from-oculus-founder-palmer-luckey/, September 2020.

[35] E. McCallister, T. Grance, and K. Scarfone. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). Technical Report NIST Special Publication (SP) 800-122, 2010.

[36] M. R. Miller, F. Herrera, H. Jun, J. A. Landay, and J. N. Bailenson. Personal Identifiability of User Tracking Data during Observation of 360-degree VR Video. *Scientific Reports*, 2020.

[37] H. Mohajeri Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Y. Huang, N. Feamster, E. W. Felten, P. Mittal, and A. Narayanan. Watching You Watch: The Tracking Ecosystem of Over-the-Top TV Streaming Devices. In *ACM CCS*, 2019.

[38] Mozilla Corporation and Individual mozilla.org contributors. Privacy & security guide: Oculus quest 2 vr headset. https://foundation.mozilla.org/en/privacynotincluded/oculus-quest-2-vr-headset/, November 2020.

[39] Mozilla Corporation and Individual mozilla.org contributors. What is fingerprinting and why you should block it. https://www.mozilla.org/en-US/firefox/features/block-fingerprinting/, 2021.

[40] H. Nissenbaum. *Privacy in Context - Technology, Policy, and the Integrity of Social Life*. 2010.

[41] Oculus. A single way to log into oculus and unlock social features. https://www.oculus.com/blog/a-single-way-to-log-into-oculus-and-unlock-social-features/, August 2020.

[42] Oculus. Supplemental oculus data policy. https://www.oculus.com/legal/privacy-policy/, October 2020.

[43] Oculus. Track your fitness in vr with oculus move. https://support.oculus.com/move/, 2021.

[44] Oculus Blog. Testing In-Headset VR Ads. https://www.oculus.com/blog/testing-in-headset-vr-ads, Sep 2021.

[45] Ole André V. Ravnås. Frida - dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. https://frida.re/, 2021.

[46] Pi-hole. Pi-hole: Network-wide ad blocking. https://pi-hole.net/, 2021.

[47] P. A. Rauschnabel, J. He, and Y. K. Ro. Antecedents to the Adoption of Augmented Reality Smart Glasses: A Closer Look at Privacy Risks. *Journal of Business Research*, 92:374–384, 2018.

[48] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In *NDSS*, 2018.

[49] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *IMC*, 2019.

[50] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *MobiSys*, 2016.

[51] F. Roesner, T. Kohno, and D. Molnar. Security and Privacy for Augmented Reality Systems. *CACM*, 57(4):88–96, 2014.

[52] S. Rogers. Virtual reality for good use cases: From educating on racial bias to pain relief during childbirth. https://www.forbes.com/sites/solrogers/2020/03/09/virtual-reality-for-good-use-cases-from-educating-on-racial-bias-to-pain-relief-during-childbirth/, March 2020.

[53] P. Sarnoff. The vr in the enterprise report: How retailers and brands are illustrating vr's potential in sales, employee training, and product development. https://www.businessinsider.com/virtual-reality-for-enterprise-sales-employee-training-product-2018-12, December 2018.

[54] A. Shuba, A. Le, E. Alimpertis, M. Gjoka, and A. Markopoulou. AntMonitor: A System for On-Device Mobile Network Monitoring and its Applications. *arXiv preprint arXiv:1611.04268*, 2016.

[55] A. Shuba and A. Markopoulou. NoMoATS: Towards Automatic Detection of Mobile Tracking. In *PETS*, 2020.

[56] A. Shuba, A. Markopoulou, and Z. Shafiq. NoMoAds: Effective and Efficient Cross-App Mobile Ad-Blocking. In *PETS*, 2018.

[57] Y. Shvartzshnaider, N. Apthorpe, N. Feamster, and H. Nissenbaum. Going Against the (Appropriate) Flow: a Contextual Integrity Approach to Privacy Policy Analysis. In *HCOMP*, 2019.

[58] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu. Toward a Framework for Detecting Privacy Policy Violations in Android Application Code. In *ACM/IEEE ICSE*, 2016.

[59] Software Freedom Conservancy. Seleniumhq browser automation. https://www.selenium.dev/, 2021.

[60] Spatial Systems, Inc. Spatial: Virtual spaces that bring us together. https://spatial.io/, 2021.

[61] StackExchange. Micro Average vs Macro average Performance in a Multiclass classification setting. https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin, 2017.

[62] UCI Networking Group. OVRseen project page. https://athinagroup.eng.uci.edu/projects/ovrseen/.

[63] Unity. The Virtual Room ad: a real way to make money in VR. https://create.unity3d.com/virtual-room-ad, 2021.

[64] Unity Technologies. mbed tls: An open source, portable, easy to use, readable and flexible ssl library. https://github.com/Unity-Technologies/mbedtls, 2021.

[65] Unity Technologies. Unity - the leading platform for creating interactive, real-time content. https://unity.com/, 2021.

[66] Unity Technologies. Unity manual: Managed code stripping. https://docs.unity3d.com/Manual/ManagedCodeStripping.html, 2021.

[67] J. Varmarken, H. Le, A. Shuba, A. Markopoulou, and Z. Shafiq. The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking. In *PETS*, 2020.

[68] X. Wang, X. Qin, M. Bokaei Hosseini, R. Slavin, T. D. Breaux, and J. Niu. GUILeak: Tracing Privacy Policy Claims on User Input Data for Android Applications. In *IEEE/ACM ICSE*, 2018.

[69] B. Wassom. *Augmented Reality Law, Privacy, and Ethics: Law, Society, and Emerging AR Technologies*. 2014.

[70] B. Wolford. What is gdpr, the eu's new data protection law? https://gdpr.eu/what-is-gdpr/, 2019.

[71] S. Zimmeck and S. M. Bellovin. Privee: An Architecture for Automatically Analyzing Web Privacy Policies. In *USENIX Security*, Aug. 2014.

[72] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. M. Sadeh, S. M. Bellovin, and J. R. Reidenberg. Automated Analysis of Privacy Requirements for Mobile Apps. In *NDSS*, 2017.

APPENDICES

A Data Privacy on Oculus

In this appendix, we provide more details from our observations on data collection practices on OVR that complements our explanation in Section 2. In our preliminary observation, we discovered two findings that motivate us to further study VR privacy in the context of OVR.

First, we discovered that Oculus has been actively updating their privacy policy over the years. We collected different versions of Oculus privacy policy [42] over time using Wayback Machine and examined them manually. Most notably, we observed a major change in their privacy policy around May 2018. We suspect that this is due to the implementation of the GDPR on May 25, 2018 [70]—this has required Oculus to be more transparent about its data collection practices. For example, the privacy policy version before May 2018 declares that Oculus collects information about "physical movements and dimensions". The version after May 2018 adds "play area" as an example for "physical movements and dimensions". Although it has not been strictly categorized as PII, "play area" might represent the area of one's home—it loosely contains "information identifying personally owned property" that can potentially be linked to an individual [35]. This motivates us to empirically study the data collection practices on OVR. We report how we use OVRSEEN to collect network traffic and study data exposures on OVR in Section 3.

Second, we found that many apps do not have privacy policies. Even if they have one, we found that many developers neglect updating their privacy policies regularly. Many of these privacy policies even do not have *last updated times* information. We found that only around 40 (out of 267) apps from the official Oculus app store and 60 (out of 1075) apps from the SideQuest app store have updated their privacy policy texts in 2021. Thus, we suspect that an app's actual data collection practices might not always be consistent with the app's privacy policy. This motivates us to study how consistent an app's privacy policy describes the app's actual data collection practices. We report how we use OVRSEEN to analyze privacy policies in Section 4.

B Network Traffic Collection Details

In this appendix, we provide more details on OVRSEEN's system for collecting and decrypting network traffic, introduced in Section 3.1. Fig. 8 depicts a detailed version of the network traffic collection system (*i.e.,* ① in Fig. 1), which consists of two main components—AntMonitor and Frida. In Appendix B.1, we describe the improvements we made to AntMonitor; and in Appendix B.2, we provide the detailed workflow of our automated binary analysis technique for finding addresses of certificate validation functions so that we can hook into them with Frida.

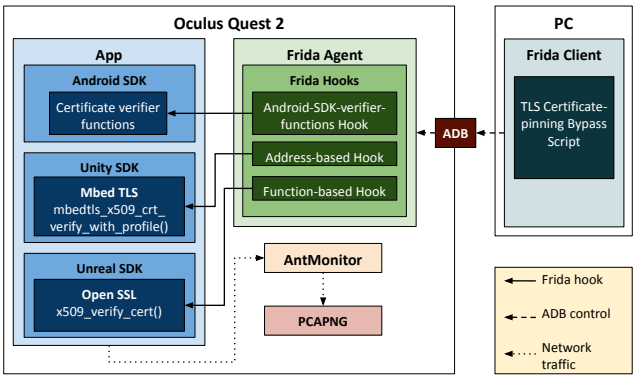


Figure 8: Network traffic collection and decryption.

B.1 Improving AntMonitor

The original version of AntMonitor has several limitations, which we address in this paper. First, the released version of AntMonitor supports only up to Android 7. Unfortunately, Quest 2 runs Oculus OS that is based on Android 10—a version of Android which underwent a multitude of changes to TLS [18] and filesystem access [20], effectively breaking AntMonitor's decryption and packet-to-app mapping capabilities. To restore decryption, we first downgraded all connections handled by AntMonitor to TLS 1.2 so that we can extract servers' certificates, which are encrypted in TLS 1.3 (the default TLS version in Android 10). The servers' certificates are needed so that AntMonitor can sign them with its own CA and pass them on to the client app. In addition to downgrading the TLS version, we also updated to new APIs for setting the SNI (server name identification), since the original version used Java reflection to access hidden methods which were no longer available in Android 10. Further, we updated how AntMonitor checks for trusted certificates to remain compatible with Android 10's stricter security requirements. Similarly, in order to fix the packet-to-app mapping, which relied on reading the now-restricted `/proc/net` files, we re-implemented the functionality using Android 10's new APIs from the `ConnectivityManager`.

Second, AntMonitor prevents common traffic analysis tools, such as `tshark`, from re-assembling TCP streams because it saves both encrypted and decrypted versions of packets in the same PCAPNG file and does not adjust the sequence and ack numbers accordingly. In our work, we wanted to take advantage of `tshark`'s re-assembly features, namely the desegmentation of TCP streams and HTTP headers and bodies, so that we could analyze parsed HTTP traffic with confidence. To that end, we modified AntMonitor to keep track of decrypted sequence and ack numbers for each decrypted flow and to save decrypted packets in a separate PCAPNG file with their adjusted sequence and ack numbers. Without this improvement, the encrypted and decrypted packets would share the same sequence and ack numbers, inhibiting TCP re-assembly.

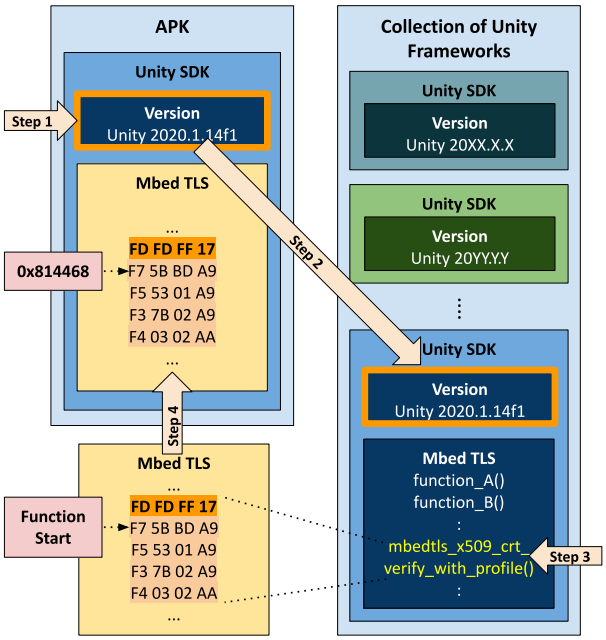


Figure 9: **Our decryption technique.** Example on Spatial, an app that enables people to meet through VR [60].

In order to enable other researchers to continue using AntMonitor in newer Android versions, we will submit a pull request to its open source repository.

B.2 Binary Analysis Workflow

In Section 3.1 we introduced our automated binary analysis technique for finding addresses of certificate validation functions in Unity's [64] library so that we can hook into them with Frida. Fig. 9 illustrates how this technique is applied on Spatial, an app that enables people to meet through VR [60], as our example. First (**Step 1**), we take the app's APK file and extract the version of the Unity framework used to package the app by scanning its configuration files—here we find that the Spatial app uses Unity 2020.1.14f1. Second (**Step 2**), we try to find `Unity 2020.1.14f1` from a collection of Unity frameworks (we first have to download all versions of Unity onto our system). Third (**Step 3**), we locate `mbedtls_x509_crt_verify_with_profile()` in the (non-stripped) symbolicated pre-compiled Unity SDK binary file that comes with `Unity 2020.1.14f1`. Subsequently, we extract the binary signature of the certificate validation function, which consists of the 4 bytes preceding the start of the function (*i.e.,* **`FD FD FF 17`**) and the first 16 bytes starting from the function address (*i.e.,* `F7 5B BD A9 F5 53 01 A9 F3 7B 02 A9 F4 03 02 AA`). We found that we could not use the entire function as our signature due to binary compilation optimizations and stripping. Fourth (**Step 4**), we use this binary signature to locate `mbedtls_x509_crt_verify_with_profile()` in the app's stripped binary file and extract its actual address—for the Spatial app the function is located at address **`0x814468`**. Finally, we use this extracted address to set a Frida hook for `mbedtls_x509_crt_verify_with_profile()` in the Frida script (see Fig. 8).

C Data Types and ATS Details

In Appendix C.1, we provide details about how we identify and group data types, which complements our work in Section 3.2.2. In Appendix C.2, we provide the full list of potential ATS destinations that are missed by blocklists, which complements our work in Section 3.3.

C.1 Extracting Data Types

Please recall that Section 3.2.2 introduced our methodology for extracting data types from our network traffic dataset.

Data types can be identified through static values (*e.g.,* Email, Serial Number, Android ID) which rely on string matching of keywords. On the other hand, dynamic values can change based on the application being tested (*e.g.,* SDK Version), which rely on a combination of string matching and regular expressions. Table 6 provides the details on the keywords and regular expressions that we use to extract data types. For instance, to capture different versions of Unity SDK Versions being exposed, we rely on the regular expression `UnityPlayer/[\d.]+\d`.

Our 21 data types are groups of other finer grain data types, detailed in Table 6. For example, the data type SDK Version considers both Unity and Unreal versions, while Usage Time considers the Start Time and Duration of app usage. Grouping of data types allows us to provide a more complete picture of data collection on OVR.

C.2 Missed by Blocklists

As OVR is an emerging platform, there are currently no specialized blocklists for it. To facilitate the identification of domains that are potential ATS, we target domains that collect multiple different data types. As a result, we extend Table 2 from Section 3.3 and provide the full details of domains that were missed by blocklists in Table 7.

D Privacy Policy Analysis Details

In this appendix, we provide more details about OVRSEEN's privacy policy analysis we described in Section 4. We describe the details of our improvements for PoliCheck in Appendix D.1, our manual validation for PoliCheck in Appendix D.2, and how we integrated Polisis into OVRSEEN (including our manual validation for Polisis) in Appendix D.3.

PII	Finer Grain Data Types	Keywords or `Regular Expressions`
Android ID	-	(hard-coded Android ID), android_id, x–android–id
Device ID	-	(hard-coded Oculus Device ID), deviceid, device_id, device–id
Email	-	(hard-coded user email address), email
Geolocation	Country Code, Time Zone, GPS	countryCode, timeZoneOffset, gps_enabled
Person Name	First Name, Last Name	(hard-coded from Facebook Account)
Serial Number	-	(hard-coded Oculus Serial Number), x–oc–selected–headset–serial
User ID	User ID and PlayFab ID	user_id, UserID, x–player, x–playeruid, profileId, anonymousId, PlayFabIDs
Fingerprint		
App Name	App Name and App Version	app_name, appid, application_name, applicationId, X–APPID, gameId, package_name, app_build, localprojectid, android_app_signature, gameVersion, package_version
Build Version	-	build_guid, build_tags
Cookies	-	cookie
Flags	Do Not Track, Tracking, Jail Break, Subtitle On, Connection Type, Install Mode, Install Store, Scripting Backend	x–do–not–track, tracking, rooted_or_jailbroken, rooted_jailbroken, subtitles, connection–type, install_mode, install_store, device_info_flags, scripting_backend
Hardware Info	Device Model, Device RAM, Device VRAM, CPU Vendor, CPU Flags, Platform CPU Count and Frequency, GPU Name, GPU Driver, GPU Information, OpenGL Version, Screen Resolution, Screen DPI, Fullscreen Mode, Screen Orientation, Refresh Rate, Device Info, Platform	device_model, device_type, enabled_vr_devices, vr_device_name, vr_device_model, Oculus/Quest/hollywood, `Oculus[+ ]?Quest`, `Quest[ ]?2`, device_ram, device_vram, Qualcomm Technologies, Inc KONA, ARM64 FP ASIMD AES, ARMv7 VFPv3 NEON, cpu_count, cpu_freq, ARM64+FP+ASIMD+AES, arm64-v8a,+armeabi-v7a,+armeabi, Adreno (TM) 650, GIT@09c6a36, GIT@a8017ed, gpu_api, gpu_caps, gpu_copy_texture_support, gpu_device_id, gpu_vendor_id, gpu_driver, gpu_max_cubemap_size, gpu_max_texture_size, gpu_shader_caps, gpu_supported_render_target_count, gpu_texture_format_support, gpu_vendor, gpu_version, OpenGL ES 3.2, `\+3664`,`\+1920` , 3664 x 1920, 3664x1920, width=3664, screen_size, screen_dpi, is_fullscreen, screen_orientation, refresh_rate, device_info_flags, releasePlatform, platform, platformid
SDK Version	Unity Version, Unreal Version, Client Library, VR Shell Version	`Unity[- ]?v?20[12]\d \.\d +\.\d +`, `Unity[- ]?v?[0-6]\.\d +\.\d +`, `UnityPlayer/[\d .]+\d`, `UnitySDK-[\d .]+\d`, x–unity–version, sdk_ver, engine_version, X–Unity–Version, sdk_ver_full, ARCore, X-UnrealEngine-VirtualAgeStats, engine=UE4, UE4 0.0.1 clientLib, clientLibVersion, x–oc–vrshell–build–name
Session Info	App Session, Session Counts, Events, Analytics, Play Session Status, Play Session Message, Play Session ID	AppSession, session_id, sessionid, event-id, event_id, objective_id, event-count, event_count, session-count, session_count, analytic, joinable, lastSeen, join_channel, JoinParty, SetPartyActiveGameOrWorldID, SetPartyIDForOculusRoomID, JoinOpenWorld, partyID, worldID, gameOrWorldID, oculusRoomID
System Version	-	(hard-coded OS version strings), x-build-version-incremental, os_version, operatingSystem, os_family
Usage Time	Start Time, Duration	t_since_start, startTime, realtimeDuration, seconds_played, game_time, gameDuration
Language	-	language, language_region, languageCode, system_language
VR Sensory Data		
VR Field of View	-	vr_field_of_view
VR Movement	Position, Rotation, Sensor Flags	vr_position, vr_rotation, gyroscope, accelerometer, magnetometer, proximity, sensor_flags, left_handed_mode
VR Play Area	Play Area, Play Area Geometry, Play Area Dimention, Tracked Area Geometry, Tracked Area Dimension	vr_play_area_geometry, vr_play_area_dimension, playarea, vr_tracked_area_geometry, vr_tracked_area_dimension
VR Pupillary Distance	-	vr_user_device_ipd

Table 6: **Extracting data types.** Summarizes how we group data types and the keywords and regular expressions (italicized) that we use to identify them (Section 3.2 and Appendix C.1).

FQDN	Organization	Data Types
bdb51.playfabapi.com, 1c31b.playfabapi.com	Microsoft	Android ID, User ID, Device ID, Person Name, Email, Geolocation, Hardware Info, System Version, App Name, Session Info, VR Movement
sharedprod.braincloudservers.com	bitHeads Inc.	User ID, Geolocation, Hardware Info, System Version, SDK Version, App Name, Session Info, Language
cloud.liveswitch.io	Frozen Mountain Software	User ID, Device ID, Hardware Info, System Version, App Name, Language, Cookie
datarouter.ol.epicgames.com	Epic Games	User ID, Device ID, Hardware Info, SDK version, App Name, Session Info
9e0j15elj5.execute-api.us-west-1.amazonaws.com	Amazon	User ID, Hardware Info, System Version, SDK Version, Usage Time
63fdd.playfabapi.com	Microsoft	Android ID, User ID, Email, SDK Version, App Name
us1-mm.unet.unity3d.com	Unity	Hardware Info, System Version, SDK Version, Usage Time
scontent.oculuscdn.com	Facebook	Hardware Info, System Version, SDK Version
api.avatarsdk.com	Itseez3d	User ID, Hardware Info, SDK Version
52.53.43.176	Amazon	Hardware Info, System Version, SDK Version
kingspraymusic.s3-ap-southeast-2.amazonaws.com, s3-ap-southeast-2.amazonaws.com	Amazon	Hardware Info, System Version, SDK Version
pserve.sidequestvr.com	SideQuestVR	Hardware Info, System Version, Language
gsp-auw003-se24.gamesparks.net, gsp-auw003-se26.gamesparks.net, gsp-auw003-se30.gamesparks.net, live-t350859c2j0k.ws.gamesparks.net	GameSparks	Device ID, Flags
yurapp-502de.firebaseapp.com	Alphabet	Hardware Info, SDK Version

Table 7: **Missed by blocklists continued.** We provide third-party FQDNs that are missed by blocklists based on the number data types that are exposed. This is the full details of Table 2.

D.1 Other PoliCheck Improvements

In Section 4.1.1, we mentioned that we have improved PoliCheck in OVRSEEN. We detail the improvements below.

Inclusion of third-party privacy policies. PoliCheck assumes that each app has one privacy policy. In practice, many apps do not disclose third-party data collection clearly in the privacy policies. Instead, they put links to external third-party policies and direct users to read them for more information. For example, consider the following sentence from one of the privacy policies of apps in our dataset: *"For more information on what type of information Unity collects, please visit their Privacy Policy page <link>..."*

OVRSEEN's privacy policy analyzer includes statements from external privacy policies if they are referred to in the app's privacy policy. In this case, first-person pronouns (*e.g.,* "we") in the external privacy policies are translated to the actual entity names (*e.g.,* "Unity"). Thus, in the above example, the app's data flows are checked against the policy statements extracted both from the app's privacy policy and Unity's privacy policy.

Resolution of first-party entity names. Some privacy policies use full company names to refer to the first party, while PoliCheck only considers first-person pronouns (*e.g.,* "we") as indications of first-party references. Thus, we found that PoliCheck wrongly recognizes these company names as third parties. As a result, first-party data flows of these apps were wrongly classified as omitted disclosure.

To fix this issue, OVRSEEN privacy policy analyzer uses a per-app list of first-party names—this list was extracted from: (1) package names, (2) app store metadata, and (3) special sentences in privacy policies such as titles, the first occurrence of a first-party name, and copyright notices. These names are treated as first party.

Entities. Entities are names of companies and other organizations. We translate domains to entities in order to associate data flows with disclosures in the privacy policies in Section 4.

Similar to [5], we use a manually-crafted list of domain-to-entity mappings to determine which entity that each domain belongs to. For example, `*.playfabapi.com` is a domain of the entity Playfab. We started from the original PoliCheck's mapping, and added missing domains and entities to it. We visited each new domain and read the information on the website to determine its entity. If we could not determine the entity for a domain, we labeled its entity as *unknown third party*. Fig. 3b displays a partial view of our entity ontology.

Label	Prec.	Recall	F1	Support
three-class classification				
consistent	0.93	0.74	0.82	454
incorrect	0.50	1.00	0.67	2
omitted	0.77	0.94	0.85	425
macro-average	0.74	0.89	0.81	
micro-average	0.84	0.84	0.84	
binary classification				
inconsistent (positive)	0.77	0.94	0.85	427
consistent (negative)	0.93	0.74	0.82	454

Table 8: **PoliCheck validation.** Multi-class and binary classification metrics for each disclosure type along with the averaged performance. Note that support is in terms of number of data flows.

Label	Prec.	Recall	F1	Support
additional service feature	0.74	0.70	0.72	20
advertising	0.94	1.00	0.97	16
analytics research	0.91	0.80	0.85	25
basic service feature	0.82	0.45	0.58	20
legal requirement	0.64	1.00	0.78	9
marketing	0.92	0.75	0.83	16
merger acquisition	0.78	0.88	0.82	8
personalization customization	0.80	0.67	0.73	6
service operation and security	0.82	0.64	0.72	14
unspecific	0.75	0.90	0.81	49
macro-average	0.81	0.78	0.78	
micro-average	0.80	0.79	0.79	

Table 9: **Polisis validation.** Multi-label classification metrics for each purpose along with the averaged performance. Note that support is in terms of number of text segments. Text segments that Polisis does not annotate with a purpose is annotated as "unspecific".

D.2 PoliCheck Validation

We briefly described our manual validation for PoliCheck in Section 4.1.3. To test the correctness of OVRSEEN's privacy policy analyzer, which is based on PoliCheck that was ported into the VR domain, we followed the methodology described in the PoliCheck paper [5] and another study that applies PoliCheck on Alexa skills [29]. They sampled a portion of consistency results and manually read through the corresponding privacy policies to validate the results.

In PoliCheck, network-to-policy consistency analysis is a single-label five-class classification task. To mitigate biases from human annotators, PoliCheck authors skipped ambiguous disclosures and did not differentiate between clear and vague disclosures during manual validation, which turned it into a three-class (*i.e.,* consistent, omitted, and incorrect) classification task. We followed this validation methodology and obtained the complete results that are shown in Table 8. The authors reported micro-averaged precision [5]. For completeness and consistency with PoliCheck results, we also report recall, F1-score, and macro-averaged metrics. Micro- and macro-averaging are both popular methods to calculate aggregated precision and recall in multi-class classification tasks [61]. Macro-averaged precision/recall simply reports the averaged precision/recall of each class. For example, macro-averaged precision is

$$Pr_{\text{macro}} = \frac{1}{N}(Pr_1 + Pr_2 + ... + Pr_N)$$

where N is the number of classes and Pr_i is the precision of class i. In contrast, micro-averaging sums numbers of true positives and false positives of all classes first, and then calculates the metrics. Thus, micro-averaged precision is

$$Pr_{\text{micro}} = \frac{\text{TP}_1 + \text{TP}_2 + ... + \text{TP}_N}{(\text{TP}_1 + \text{TP}_2 + ... + \text{TP}_N) + (\text{FP}_1 + \text{FP}_2 + ... + \text{FP}_N)}$$

where TP_i and FP_i are numbers of true positive and false positive samples of class i. In single-label multi-class classification, every misclassification is a false positive for one class and a false negative for other classes. Thus, the denominators in precision and recall are always equal to the population of samples: micro-averaged precision, recall and F1-score are all the same. Micro-averaging is preferable when the distribution of classes is highly imbalanced, which is the case in our dataset.

In addition, we also report, in Table 8, the precision, recall and F1-score of the binary classification case, where we only care about whether the data flows are consistent or not with privacy policy statements. In this case, inconsistent flows are seen as positive samples.

D.3 Polisis Integration and Validation

We described how we used Polisis for purpose extraction in Section 4.2. Polisis is available as a black-box online privacy policy analysis service (https://pribot.org/). We feed privacy policies (in HTML format) into Polisis and get text segments annotated with purposes via Polisis Web API. To the best of our knowledge, it internally uses end-to-end deep learning classifiers to annotate purposes at text-segment level [21], which is different from PoliCheck's sentence-level NLP technique. Since Polisis is not open-sourced, we know very little about how Polisis segments and processes text internally.

We developed a translation layer to associate OVRSEEN data flows with purposes from Polisis. The translation layer emulates PoliCheck's text processing on Polisis text segments to break them into sentences. Next, it compares binary bag-of-words representation to match sentences from PoliCheck with sentences from Polisis. Two sentences from both sides match if one sentence contains all the words in the other sentence. A successful match yields *data type* and *destination* from

PoliCheck, and *purpose* from Polisis. Although we made the sentence matching very tolerant, it still failed to find some matches due to edge cases caused by the very different text processing pipelines of PoliCheck and Polisis.

Polisis validation. We evaluated the performance of Polisis by manually annotating text segments with purposes. The evaluation process is described in Section 4.2 and the complete results are shown in the upper part of Table 9.

E Responses from Developers

We sent courtesy notification emails to inform Oculus and the developers of the 140 apps about our findings on September 13 and 14, 2021. We provide a summary of responses from these developers in Section 5.2. Within a period of two weeks, we received 24 responses from these developers: three developers of Oculus free apps, six developers of Oculus paid apps, and 15 developers of SideQuest apps. Most of these developers (21/24) responded positively and thanked us for sharing our findings about their apps; others responded simply that they have received the message (*e.g.,* through an automated reply), or said that the email address we sent our message to was the wrong one. Five of 19 developers reiterated their position about their data collection practices and/or referred us back to their privacy policy. Notably, 12 of 19 developers inquired further about our findings: they discussed with us to gain deeper insights from our findings, promised to improve their privacy policy, and asked for our advice on how they can write better privacy policies. In particular, some developers expressed the need for training on privacy policy writing and the difficulty in ensuring consistent disclosures—this implicates the need for tools, such as OVRSEEN.