Design of Privacy Preservation System in Augmented Reality

Yoonsang Kim, Saeed Boorboor, Amir Rahmati, and Arie E. Kaufman, *Fellow, IEEE* *

Stony Brook University

Abstract

Augmented Reality (AR) capability to overlay virtual data on top of real-world objects and enable better understanding of visual inputs attract the attention of application developers and researchers alike. However, the privacy challenges associated with the use of AR systems is not sufficiently recognized. We present Erebus, a privacy-preserving framework designed for AR applications. Erebus allows the user to establish fine-grained control over the visual data accessible to AR applications. We explore use cases of Erebus framework and how it can be applied to safeguard the privacy of the user's surroundings in AR environments. We further analyze the latency penalty imposed by Erebus to understand its effect on user experience.

Keywords: Augmented Reality, Augmented Reality Security, Privacy Preservation, Collaborative Augmented Reality, Unity

Index Terms: Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Mixed / augmented reality; Security and privacy—Software and application security—Social network security and privacy

1 Introduction

The field of Augmented Reality has experienced unprecedented growth in recent years. This growth has led to an increasing concern regarding the security and privacy of these platforms [4]. There has been multiple attempts to manage the privacy challenges of AR applications. The work by Hu et al. [5] suggests the need to access-control the permission of AR applications on the client's device. This work isolates the permission of each installed application by adding a control layer within the client's device. This way, the system can enforce privacy-safe data communication between the applications and the system of device. The work by David-John et al. [3] proposes an approach to cipher the gaze data that are commonly used in many AR applications. This work suggest two approaches: (1) Adding a privacy safeguarding layer between the application and the operating system of a device, and (2) Resampling/adding noise to the original data. The latter method is typically used for applications that require full access to the original data. As these two works show, AR applications suffer from 'All-or-nothing' property. They need a complete access to the user's surroundings in order to overlay data on top of physical world objects. To protect users from unwillingly disclosing their privacy, we introduce Erebus. Erebus focuses on the privacy of camera inputs on client's device and enables fine-grained control over the camera inputs accessible by the application. Our insight is that, to achieve their functionality, most AR applications only need access to a subset of components in the field of vision. Erebus works by obscuring the parts of input camera superfluous to application, allowing external applications to access only a subset of camera input required for their functionality. This is achieved by creating a permission model that allows applications to define their required data, establishing a privacy preservation policy for the camera data.

*E-mail:{ yoonsakim,sboorboor, amir, ari} @cs.stonybrook.edu

2 Problem Statement

We suggest a scenario where a serious threat can be introduced. Any two or more clients with AR applications connected through network. An AR video chat application is a good example to illustrate the case. The application which heavily relies on real-time video data synchronization and communication.

Scenario: There is a possibility of a threat actor collecting data in between network-connected clients or eavesdropping data as a malicious remote client. To be specific, the malicious actor may obtain the private data of clients from their video camera inputs. However, we assume that the integrity of internal network security for the sender client is not compromised. In other words, the threat can only exist on two locations: The external network path of clients and at the receiving remote client-side.

3 System Design

Our base approach to solve the privacy exposure is by letting the sender clients process the privacy-sensitive information themselves. As illustrated in Figure 1, the client system is responsible for obscuring any privacy-sensitive data on its side before any data leaves the client's AR device. This approach can readily solve our threat scenario because the only data the attacker can acquire is the obscured data.

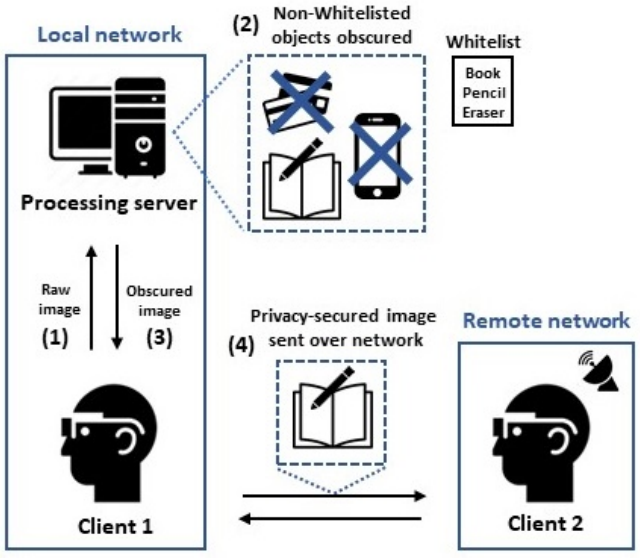

Figure 1: Architectural overview of Erebus and its four main steps

Whitelisting: As depicted as Step 2 in Figure 1, a client maintains a list of object names to be classified as 'Public' type objects. Objects that are not contained in this list are obscured and considered as 'Private' type data. We chose to utilize the concept of 'Whitelisting' over 'Blacklisting' approach because Blacklisting requires full annotation and complete understandings of the client's surroundings. It is vulnerable to unseen objects for the object classification model. Unlike the Whitelisting approach which views any unidentified objects as Private type objects, the unseen objects will recognize as Public type data and be displayed to other clients. In a real-world scenario, stationary objects with not only

Figure 2: Concept visualization of three different privacy preservation methods: Whitelisting (Left), Blacklisting (Middle), Whitelisting with instance segmentation (Right). These conceptual figures were generated using YOLACT++ [2]

clear boundaries, but also vaguely-defined boundaries such as a wall, a parking lot, and scenery exist. Thus, it is difficult to maintain a list of all the blacklist objects of an input scene. Moreover, failing to obscure the background scene of an input may provide enough context to a malicious actor to infer a few missing data. The concept visualization of Whitelisting and Blacklisting approaches are shown in Figure 2.

Data Obscuring: We obscure user's data before it is transmitted to a remote client. Our data obscure method is performed on bounding-box of each object. To achieve this, we utilize deep learning-based object detection model. The raw camera input from an AR device is fed into the object detection model and its classified result is compared with the whitelist. If the object is classified as Private, we obscure the object so that is safeguarded from being displayed to a remote client. Any surrounding environment displayed in the raw input data is masked out except the Public type objects. This way, the clients are able to control their privacy and keep their private data within their local systems while publicly streaming their video camera inputs to other clients.

Task Partitioning: Mobile AR devices yet lack the ability to compute heavy computer vision tasks such as executing a deep learning-based object detection model. Thus, we devised of adding a trusted device with superior computation power to the sender client's local network. This device will act as a local processing server for heavy-duty object detection allowing the AR device to concentrate only on data obscure and network communication with the remote server. This is illustrated as Processing server in Figure 1.

System Transferability: A platform-specific system is limited in execution environment flexibility. We utilize Unity [7] game engine as the client environment so that our application can be used across different AR-capable devices (e.g., Hololens, Smartphones). Also, because we receive the result of the detected objects from an external device via network, as long as the bounding box format ({label, confidence, bottom_left_pos_x, bottom_left_pos_y, bbox_width, bbox_height}) is consistent, the object detection model can easily be replaced to another.

4 Implementation

This section elaborates on the concepts and the algorithms mentioned in Section 3 and their implementation details.

Local Processing Server: For the sake of responsive real-time AR application, we utilize a one-stage object detector, YoloV4 [1]. Once a JPG formatted image is received from the client, it detects objects and sends the bounding box of each object to the client.

Client: Client extracts the camera input and encodes it to a JPG format to transmit to the local processing server. When the result returns back from the local server, the client integrates the bounding box data with the current frame of the camera input.

Networking: We use Python ZeroMQ [6] for our networking module. It supports both C# and Python and we use its TCP/IP protocol to interact with the processing server and the client.

5 Performance Analysis

We analyze the latency of each step to identify the performance bottleneck of the application. Our study, shown in Table 1, indicates that the most amount of time was consumed in the Networking step followed by the Object detection step. The remaining steps contributed 4.3% (Mean:13.4ms) to the total consumed time. The time consumed on Whitelisting step was negligible as it is simple string comparisons of returned results from the object detection model. Our test was performed under the TCP protocol, wireless network (Download:16.18Mbps, Upload:21.61Mbps), and consumer-level server hardware specs (NVIDIA GTX 960, Intel i5-8400, 16GB).

Table 1: Time consumption analysis per step (ms)

Trial	COM	DCOM	NET	DET	PAR	WL	TOT
1	10	2	170	130	1	<1	313
2	10	3	158	125	1	<1	297
3	7	2	160	129	1	<1	299
4	12	4	171	126	<1	<1	313
5	9	3	169	126	1	<1	308

COM: Image compression; DCOM : Image decompression;
NET: Time spent on networking–send/recv–with processing server;
DET : Object detection/classification; PAR : Parse object detection result
WL: Time spent on Whitelisting; TOT: Total consumed time

6 Experiments

We tested the performance of mobile version of YoloV4, YoloV4-tiny, on an AR-capable mobile device (Samsung Galaxy S8) using Unity Barracuda. There was a significant gain in the speed, but the trade-off between the detection accuracy was not negligible. This was expected result according to the speed/accuracy comparison by Bochkovskiy et al. [1], YoloV4-tiny is 62% less accurate than YoloV4 detection model. Since it is crucial for our framework to correctly whitelist the detected objects, we tentatively crossed out this option in this study.
Another experiment was modifying the privacy preservation method. Instead of using solid black color to obscure information, we blurred the original input by applying a box filter and a noise texture. However, this required multi-pass rendering, adding an additional latency.

7 Conclusion and Future Work

We proposed our preliminary research, Erebus, a framework to safeguard a user's privacy in AR environment. Our concept accentuates the importance of privacy preservation in AR applications and presents a crucial element to be considered in the AR development amidst the recent upswing trend of AR. For future works, we plan to improve our framework so that it can be applicable to real-time AR applications with acceptable magnitude of latency. Also, we expect to develop example applications to demonstrate our concept.

Acknowledgments

This work was supported by the Office of Naval Research Award No. N00014-20-1-2858.

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint:2004.10934*, 2020.

[2] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee. Yolact++: Better real-time instance segmentation. *arXiv preprint arXiv:1912.06218*, 2019.

[3] B. David-John, D. Hosfelt, K. Butler, and E. Jain. A privacy-preserving approach to streaming eye-tracking data. *IEEE Transactions on Visualization and Computer Graphics*, 27(5):2555–2565, 2021.

[4] J. A. De Guzman, K. Thilakarathna, and A. Seneviratne. Security and privacy approaches in mixed reality: A literature survey. *ACM Computing Surveys (CSUR)*, 52(6):1–37, 2019.

[5] J. Hu, A. Iosifescu, and R. LiKamWa. Lenscap: split-process framework for fine-grained visual privacy control for augmented reality apps. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 14–27, 2021.

[6] iMatix. *ZeroMQ*, July 2021.

[7] Unity Technologies. *Unity*, July 2021.