UNIVERSITY OF CALIFORNIA,
IRVINE

Visor: A System Solution for Trustworthy and Effective Reports for Social VR

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Science

by

Kyoungwon Kim

2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my research advisor and chair of my committee Prof. Ardalan Amiri Sani for inspiring this work, and guiding me through this process.

I would like to thank my committee members Prof. Marco Levorato and Prof. Qi Alfred Chen.

I would like to acknowledge the funding support made by the National Science Foundation (NSF) Award #1718923.

Last but not least, I thank my collaborator Meta Novitia for working on the project and conducting the large-scale study with me.

ABSTRACT OF THE THESIS

Visor: A System Solution for Trustworthy and Effective Reports for Social VR

By

Kyoungwon Kim

MASTER OF SCIENCE in Computer Science

University of California, Irvine, 2020

Assistant Professor Ardalan Amiri Sani, Chair

Social VR apps, which enable users to interact in virtual spaces, are increasingly popular. Unfortunately, a large number of harassment incidents have taken place in these apps. To mitigate this issue, VR platforms and apps ask victims to report the incidents to analyze the reports, and if needed, punish the harassers. However, existing reporting solutions are ad hoc and cannot provide strong guarantees for integrity and authenticity of reports. Also, reports cannot always identify the harassers. To address these problems, we introduce Visor, a system solution for providing trustworthy and effective reports for social VR. To enable trustworthy reports, Visor incorporates a secure recorder on VR headsets to capture tamper-proof and irrefutable snapshots of visual content shown on the headset's display. To enable effective reports, Visor introduces mandatory recording, a novel paradigm, which, alongside a search engine in an auditing server, facilitates identifying the harasser in a report. Our security analysis and evaluation with a VR headset prototype show that Visor is secure and adds small performance overhead. Moreover, using a 12-hour 99-user social VR experiment with automated users, we demonstrate that the mandatory recording paradigm helps an auditor find a harasser with 4 minutes of manual effort, on average.

Chapter 1

Introduction

Virtual Reality (VR) is increasingly popular. It is predicted that around 50 million VR devices will be shipped in 2022 (up from 8 million devices in 2018) [4]. Moreover, users' spending on VR apps and content is expected to reach about \$3 billion, up from \$750 million in 2017 [29]. In general, the trends indicate that VR will form an important part of users' digital experience in the coming years.

An increasingly important and popular category of VR apps is social VR, which allows users to interact with each other in *virtual spaces* [35]. Social VR mainly refers to apps dedicated to social interactions. It can also refer to other VR apps (e.g., games and live sports events), which incorporate social interactions. Some examples of popular social VR apps are AltSpaceVR, High Fidelity VR, Rec Room, VR chat, and vTime. Examples of virtual spaces created by these apps are concerts, comedy clubs, discussion rooms, recreation rooms, and dance parties [35]. It is predicted that VR will be the future of social networks [35] due to its immersive experience and hence social VR apps will only grow in popularity going forward.

While social VR is relatively new (about 5 years old),an unfortunate problem has arisen: *harassment*. A recent survey of 609 users, who use VR headsets at least twice a month, revealed disturbing results [31, 32]. According to the survey, more than half of the respondents reported having experienced or witnessed, in social VR apps, some form of sexual harassment including (but not limited to) offensive gestures, groping, blocking, slapping, inappropriate touching, and lewd drawings and images [31, 32, 33]. There have been other reports of harassment in VR apps as well [22]. In fact, victims of these incidents have described the harassment to "[feel] just as real" [22], showing that VR harassment cannot be ignored simply because it happens in a virtual space and not the real world. The VR industry is aware of this important problem and has taken some steps to address it. A key solution is *reporting*. That is, some VR platforms and apps allow a victim to report an incident, after which they analyze the report and issue punishment to identified harassers. For example, Oculus allows users to report harassment incidents, either by submitting a textual report or by recording a video of the incident [25].

Unfortunately, existing reporting solutions have two important shortcomings. First, the reports are not trustworthy. That is, the integrity and authenticity of reports cannot be verified. This can result in either an authentic report being ignored or a fake report being used to frame a user. Second, the reports cannot always identify the harasser so that they can be punished.

To address these limitations, we present Visor, a first-of-its-kind system to enable *trustworthy and effective reports* for social VR. Visor has two key components. The first component is a *secure recorder* that allows capturing tamper-proof and irrefutable snapshots of content seen on the VR headset by a user. These snapshots can be used as trustworthy evidence in the report since an auditor can verify their integrity and authenticity. Using a secure recorder requires no support from a VR app. A user can simply enable it on their device and use it to generate trustworthy reports. Therefore, we believe it is a practical design that can significantly improve the state of the art of reporting in social VR.

Despite its benefits, trustworthy reports cannot guarantee to identify a harasser (needed to impose punishments). This is because they might only capture the avatar of the harasser in a snapshot but cannot link this avatar to the virtual identity of the harasser (i.e., a unique ID for the harasser). Therefore, we introduce *mandatory recording*, a novel paradigm in Visor that mandates users of a social VR app to continuously upload snapshots captured by Visor's secure recorder to an auditing server as long as they are using the app. When a victim reports an incident, these snapshots can be searched to find the harasser. We note that the mandatory recording paradigm requires some support from the social VR app. It also requires an auditing server to collect the snapshots. Therefore, we envision that only select social VR apps that intend on creating a completely safe environment for their users to adopt this paradigm. Indeed, mandatory recording can be adopted for just select virtual spaces within an app.

We answer two research questions in this thesis.

Q1. How can the secure recorder capture tamper-proof and irrefutable snapshots of visual content on the VR headset display without incurring significant performance overhead? We introduce a software-only secure recorder that leverages a hypervisor (in order to provide a small Trusted Computing Base (TCB)) as well as a Trusted Platform Module (TPM) to achieve these properties. The hypervisor uses a novel technique, called *GPU DMA command side-loading*, to capture the framebuffer without incurring significant performance overhead. Moreover, the TPM helps us provide integrity and authenticity proofs for the snapshots.

Q2. In the mandatory recording paradigm, how can the snapshots submit by all users of a social VR app be efficiently searched to find a harasser? Manually searching a large number of snapshots to find the harasser is non-trivial. To facilitate this, we introduce a location-based index for the snapshots, computed when they are submitted. Upon receiving a report, the auditor issues search queries based on this index in order to find promising snapshots. The auditor reviews these snapshots until they see the harassment from the point of view of the harasser. Once found, the auditor easily finds the virtual identity of the harasser as snapshots are digitally signed and linked to the virtual identity of their user.

We design Visor's secure recorder so that it can be deployed on today's VR headsets. To demonstrate this, we deploy the recorder for the HTC Vive Pro tethered headset without having access to the source code of the VR framework or the internals of the headset hardware. The design of the secure recorder is applicable to other types of headsets as well including standalone headsets.

We report extensive evaluation of Visor. First, we report a security analysis for our design and show that Visor can defeat various attacks that attempt to circumvent its guarantees. Second, we evaluate the secure recorder and show that it adds a small overhead to the performance of the VR app. Finally, we evaluate the effectiveness of the mandatory recording paradigm by conducting a large-scale social VR experiment with 99 automated users using AltSpaceVR for about 12 hours. We program the automated users to mimic realistic users in social VR apps and collect 413,368 snapshots from their devices. Using this large dataset, we demonstrate that our indexing and search algorithms can help the auditor find the harassers with 4 minutes of manual work, on average. We will release the prototype of Visor as well as our dataset for other researchers to use and build on.

Chapter 2

Motivation & Background

In this chapter, we describe existing reporting solutions in VR and discuss their shortcomings.

We note that VR platforms and apps have deployed other solutions for harassment as well. For example, some VR apps implement a *personal bubble space*, which prevents other avatars from getting too close to one's avatar [34]. As another example, some apps enable the users to block other users and make problematic avatars disappear from their view [8, 5, 1]. These solutions require elaborate implementations in VR apps and do not prevent all types of harassment including an offensive drawing in a virtual space [33]. Therefore, in this thesis, we focus on reporting as a comprehensive solution.

2.1 Current Harassment Report Solutions

Platform-level harassment report solutions. The two most popular VR platforms today are Oculus and SteamVR. Currently, only Oculus provides a reporting solution [12]. Any offensive, harmful, illegal, or personal content and behavior that are against the Oculus Code of Conduct can be reported. When a report is filed and confirmed by the Oculus team, the reported person can get warning emails first. When reports are filed against the same user repeatedly, the reported person's Oculus account may get a suspension from one to thirty days. The Oculus account may even be permanently disabled due to repeated or egregious offenses in the worst case [15]. When one wants to report another user, they can choose either to file the report within the headset or through the harassment report web form [12, 13]. The harasser's Oculus account name (acting as the user's virtual identity here), a reason for the report, the name of the application where it happened, approximate time of the incident, and video of the incident are asked to be provided.

App-level harassment report solutions. Several popular social VR apps provide reporting solutions. AltspaceVR users can file reports either in-game or in a web form. They can report actions that violate AltspaceVR Community Standards [10]. Another example that has both an in-game and web form report system is VRChat [19]. A user can report any violations of the Community Guidelines or the Terms of Service. It includes different types of harassment including abusive sound/visual effects, crashing, or other harmful activities. Another example of a reporting system at the app level is RecRoom [17]. When a report is filed, one needs to specify the action that is against the RecRoom Code of Conduct but does not need to provide any evidence [16]. Then the reported person can be banned from the game for 24 hours.

2.2 Limitations of Current Reporting Systems

As can be seen, the reporting solutions deployed by VR platforms and apps are ad hoc, at best. Below, we identify two key shortcomings of these solutions.

Limitation 1: Weak evidence. Except for the RecRoom application, all other available report solutions ask for visual evidence, either screenshots or videos, when reports are submitted. Oculus and other applications rely on the provided evidence to make decisions to warn or punish reported users. However, unfortunately, they do not seem to be able to verify whether the reported evidence is not fabricated or falsely reported. Also, they do not seem to have a way to verify whether the reported harasser is indeed guilty.

These shortcomings may result in correct reports being ignored (due to lack of certainty in the evidence). They can also result in false accusations of innocent users. To alleviate the latter, Oculus accepts the appeal of the Code of Conduct violation [14]. However, an innocent framed user has to be able to prove that they were not the actual harasser. In practice, it is difficult to have strong proof unless the framed user has a recorded video of their game-play at the reported time. As a real-life example of false accusations, one Reddit post shares an experience with the RecRoom report system [7]. The user describes that he warned two other players that he would report them for cheating. Instead, these other players falsely reported this user. The result was that this innocent user was banned from the game for 24 hours without any violations against the Code of Conduct of the game. This shows how a weak report system can be abused.

Limitation 2: Incomplete report. Even when snapshots or videos of the incident are available in the report (and assuming that they are authentic), they cannot always reliably reveal the virtual identity of the harasser. This is because a harasser's avatar might be a generic one, which cannot be linked to a specific user. For example, VRChat recommends providing a more detailed description along with evidence when filing a "useful" report [18]. The evidence they ask for includes output logs, screenshots that contain user name tag, minimum of two minutes of video, screenshot of the "View Author" page for the reporting user, and the description of the incident. However, a victim might not be able to capture a screenshot or video of the harasser with the harasser's name tag in it.

Chapter 3

Overview

We present a system solution, called Visor, for providing trustworthy and effective reports in social VR. Visor has two key components. The first is a *secure recorder* for providing integrity and authenticity guarantees for reports in social VR. This solution only requires support on the victim's VR headset and can even be deployed on commodity tethered headsets without any support from the VR platform provider. The second is a *mandatory recording* paradigm, which builds on top of our secure recorder and provides a framework for more effectively identifying a harasser when a victim files a report. This solution requires some support from the VR app. It also requires a centralized auditing server to collect snapshots from all users of the app and to enable an auditor to search the snapshots to identify the harasser.

In Visor, we use the notion of *virtual identity* to identity users, e.g., a harasser. Visor can adopt different virtual identities. In our current design, we use the unique ID of the VR headset, such as its serial number, as the user's identity. Compared to other options, such as account names and email addresses used in today's reporting solutions, this identity has two benefits. First, it is harder to create (which makes the punishment of harassers more effective). Second, it can be directly read by the secure recorder, which runs in the hypervisor.

In the next two chapters, we elaborate on the two components of Visor.

Chapter 4

Trustworthy Reports

Our goal is to enable a victim to submit a trustworthy report, i.e., one the integrity and authenticity of which can be verified by the VR platform or app that accepts the report, i.e., the auditor. Our key idea to achieve this is to use a secure recorder to capture tamper-proof and irrefutable snapshots of the content seen by the user in the VR headset. The user can then submit these snapshots as strong evidence in the report. We do focus on snapshots as the key element of our reporting system. This is because videos are simply a collection of snapshots. We specify three requirements for a secure recorder. First, it needs to have unmediated access to the VR device hardware so that it can capture a snapshot of the display content. Second, it has to have a small TCB to be secure against software attacks by the user. Finally, its presence on the user's device must be attested to the auditor.

To satisfy the first two requirements, we choose to build our secure recorder in a hypervisor running in the user's tethered machine (i.e., the workstation connected to and powering the VR headset). Hypervisors have a much smaller code base compared to OSes and hence are more secure against software attacks. Specifically, when a hypervisor is used for security purposes, it can get rid of features needed to run and schedule multiple VMs and hence have a code base no larger than a few tens of thousands of lines of code [30]. In fact, Windows Defender Guard System in Windows 10 already uses a hypervisor for security purposes [9]. To satisfy the last requirement, we use a Trusted Platform Module (TPM) in the tethered machine to generate *an attestation message*. This message can be appended to the snapshot in order to attest to presence of the recorder.

We have designed Visor's secure recorder to be deployable on existing tethered VR headsets without requiring support from VR headset vendors. We do, however, note that VR platform support is needed to integrate the recorder in standalone and smartphone-based headsets. This is because only the device vendors can program the hypervisor in ARM SoCs.

We next discuss the challenges we solved for the secure recorder.

4.1 Hypervisor-based Secure Recorder

Figure 4.1 (Left) shows the high-level design of our hypervisor-based secure recorder. The hypervisor uses the device passthrough virtualization technique [20, 27, 26, 23, 28] to assign the VR headset and the GPU in the tethered machine to the VM hosting the VR platform software. The VM's OS controls the VR headset. It also runs the GPU device driver and uses it to render the content to be shown on the display of the headset.

Key challenge. To make a copy of the content seen on the display of the headset, the recorder needs to make a copy of the *framebuffer*, which is the memory buffer containing the pixel-level information of the displayed content. The key challenge is that *the framebuffer is stored in GPU's own memory.* Tethered headsets require powerful discrete GPUs, as evident from the list of supported GPUs for Vive headsets [6], all of which are discrete. Discrete GPUs are equipped with their own high-speed memory and use it to store the framebuffer. This means that the hypervisor cannot simply make a copy of the framebuffer using the

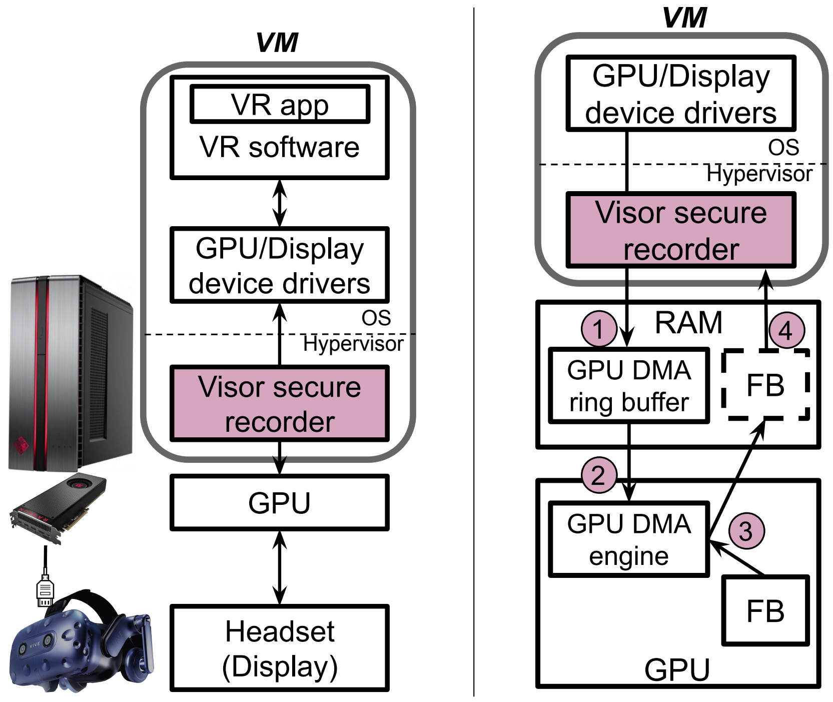

Figure 4.1: (Left) Hypervisor-based secure recorder. (Right) Secure side-loading a DMA command in Visor.

CPU's load and store instructions. It needs to find a way to transfer the framebuffer from the GPU memory to the system memory. This is, however, normally not possible since the VM controls the GPU and runs its device driver as seen in Figure 4.1 (Left).

Key idea. Our key idea is to make a copy of the framebuffer by *securely side-loading a DMA command* to the GPU, all transparent to the OS. Figure 4.1 (Right) illustrates this technique. (1) The hypervisor intercepts select interactions of the GPU driver in the untrusted OS of the VM with the GPU's DMA engine ring buffer and side-loads DMA commands to the ring buffer. (2) The GPU's DMA engine reads these commands and (3) copies the framebuffer from the GPU memory to the system memory, making it available to the recorder (4). Realizing this solution requires solving three challenges, which we discuss next.

Challenge I. Secure command injection. To side-load commands into the ring buffer, first, the hypervisor needs to know the address of the ring buffer in system memory. It finds so by monitoring the accesses of the GPU device driver to the GPU register space, specifically, when the driver writes this address to the GPU registers for the GPU to use. Second, the hypervisor needs to ensure that the OS, which controls the GPU, cannot overwrite the side-loaded commands before they are executed by the GPU. To do this, it write-protects the ring buffer memory pages just before inserting the commands and un-protects them right after the DMA is done.

Challenge II. DMA command construction. To construct the DMA command to be side-loaded to the GPU, the hypervisor needs two pieces of information: the address of the framebuffer (i.e., source buffer) and the address of the target buffer, both in the GPU address space. The hypervisor finds the address of the framebuffer by monitoring the *flip* updates by the GPU driver. Modern GPUs use double- or triple-buffering for displaying content in order to avoid tearing on the display [37]. To do so, the GPU driver constantly updates the address of the front framebuffer (as opposed to the back buffers) for the GPU to use by writing the address to designated GPU registers. By monitoring writes to these flip registers, Visor infers the address of the front framebuffer.

Finding the address of the DMA target buffer is more challenging. This is because, unlike the framebuffer, this buffer does not exist and must be created by Visor. More importantly, Visor needs to map this buffer to the GPU address space so that its address can be used by the GPU in the DMA command. To achieve this, Visor allocates several memory pages (5120 in our prototype for HTC Vive Pro's framebuffer) in the system memory and maps them in the GPU address space by updating the GPU's translation tables.

The GPU device driver maps the framebuffer in the global address space of the GPU, which is the address space used mainly by the driver but not programs. Therefore, Visor programs global translation tables (GTT) to map its target buffer to the global address space as well. The hypervisor updates multiple GTT pages, 10 pages in Visor, to map the framebuffer of a high-resolution VR headset. Before doing so, it looks for and finds unused GTT pages so that it does not overwrite those updated by the OS. Visor programs the GTT to map its target buffer in order to make its address visible to GPU for DMA commands.

Challenge III. Protecting the DMA target buffer. Visor needs to protect the DMA target buffer from the OS, which might attempt to overwrite the buffer in order to tamper with the copy of the framebuffer. The OS can potentially use two methods to write to this buffer: CPU instruction and DMA through GPU. Visor protects against the former by not mapping the buffer into the OS physical address space. Protecting against the second method is more challenging since the buffer is mapped into the GPU address space. To protect against this, Visor hides the buffer in the GPU address space. That is, for every snapshot, Visor maps the buffer into a different GPU address and unmaps the buffer after the DMA is done (and enforces a small time window for the mapping to be valid).

4.2 Attesting Presence of Recorder

In order to have confidence in the authenticity and integrity of the snapshots, one needs to be able to verify that a snapshot was indeed captured by a claimed secure recorder. We address this by using a Trusted Platform Module (TPM).

TPM's measurements. Before describing how we generate the attestation message, we describe how TPM's measurements work. TPM keeps track of the system software loaded into memory at boot time. To do this, TPM uses an append-only register called Platform Configuration Register (PCR). Every layer of the system software, e.g., BIOS and bootloader, measures the next software (i.e., calculates the hash of the code) and appends the measurement to PCR. Further, to enable attestation, TPM signs the measurements using the private key of the Attestation Identity Key (AIK). AIK is signed by the TPM's Endorsement Key (EK), which is an asymmetric key unique and identifiable to every TPM chip. Moreover, TPM allows using a 160 bit nonce in the measurement output to prove freshness. More specifically, if we show the PCR measurement as M and the nonce as n, the measurement output provided by TPM (O_{TPM}) is $\{M, n\}^{Pr_{AIK}}$.

Attestation message. Our goal is to use the TPM's measurement to generate our required attestation message. To do this, the hypervisor generates a new key pair (Pr_H, Pu_H), keeps the private key to itself (and uses to sign the snapshots), acquire the TPM's measurements output using a nonce (to prove freshness), reads the headset's unique serial number (VID, which acts as a virtual identity for the user), and generates the following message: $\{\{VID, Pu_H, O_{TPM}, Cert_{AIK}\}^{Pr_H}\}^{Pu_S}$, where Pu_S is the public key of the auditor. As can be seen, the message is signed by the private key of the hypervisor and is encrypted with the public key of the auditor.

This attestation message helps attest the required properties to the auditor. First, the auditor can verify the presence of the hypervisor that implements the recorder by inspecting the content of the TPM measurement and comparing it to an expected value. The server also checks the AIK certificate ($Cert_{AIK}$) to make sure the measurement is authentic (i.e., it is from a real TPM). Since only the hypervisor can acquire the TPM measurement, an attacker cannot acquire a valid measurement to spoof the message. Second, the auditor can verify the identity of the reporting user (i.e., the headset's unique ID) as it is included in the message. Third, the auditor can verify that the secure recorder has indeed captured a submitted snapshot. This is because, as mentioned, the secure recorder digitally signs the screenshot with Pr_H. Given that the attestation message includes the public key (Pu_H) of the hypervisor, the auditor can use this signature to verify that a snapshot has been taken by the hypervisor identified in the message.

Chapter 5

Effective Reports

The tamper-proof and irrefutable snapshots provided by Visor's secure recorder greatly improve the trustworthiness of the report. However, they cannot always help identify the harasser as the harasser's avatar in a snapshot might not reveal their identity. Therefore, we further introduce a novel paradigm, called *mandatory recording*, to facilitate finding harassers. Unlike the secure recorder, which does not require any support from the VR platform or app, mandatory recording requires small support from the VR app. Therefore, select social VR apps, which are eager to eliminate harassment, can adopt it. Indeed, a social VR app can selectively use this paradigm for some, but not all, of its VR spaces.

5.1 Workflow

Figure 5.1 illustrates the workflow of mandatory recording. The key idea is to require a VR user to first authenticate with an auditing server using their virtual identity before the user is allowed to use a social VR app. If the user is successfully authenticated by the auditing server, the server provides the user with an authentication token. The user then provides

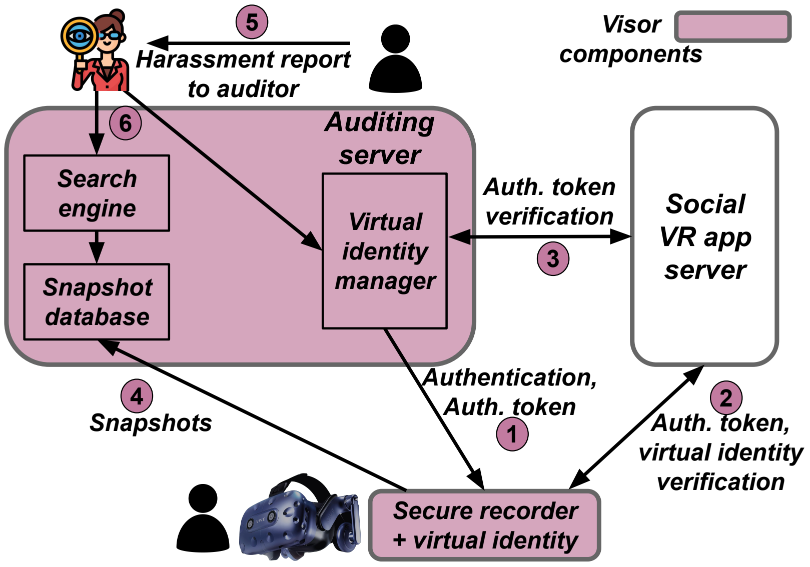

Figure 5.1: Workflow of mandatory recording.

the token to the social VR app server to gain access. The app server checks the validity of the token (by contacting the auditing server) and also verifies (through attestation) the virtual identity of the contacting user to ensure it matches the token. If successful, the user is allowed to use the app but is required to regularly (once every 2 seconds) transmit snapshots of visual content seen on the VR headset's display (and collected by Visor's secure recorder). Periodic snapshot transmission is required while the user is using the social VR app to keep the authentication token valid (hence the name, mandatory recording). If the snapshots are not transmitted at the required frequency, the user's authentication token will be revoked by the auditing server and the revocation notification is transmitted to the app server, which terminates user's access to the app.

When harassment occurs, the victim can use the same snapshots to report the incident to an auditor. The auditor then uses the search engine in the auditing server to search the snapshots from *all the users in the social VR app* to find the harasser. The goal of the auditor is to find snapshots from the harasser's own headset to corroborate the report. This is because, as mentioned, snapshots from other users at best show the harasser's avatar, which might not reveal their virtual identity. Snapshots from the harasser's own headset are however signed with a key (Pr_H) bound to their virtual identity (§4.2).

Our design for mandatory recording requires both the auditing server and the app server to verify the presence of the hypervisor on the user's machine and to verify the virtual identity of the user. This can be done using the same attestation message introduced in §4.2, albeit with one difference. For attestation in this paradigm, the auditing server and the app server sends a nonce to the secure recorder, which uses it to pass to TPM to be included in the measurement output. In §7, we discuss how this can help prevent an attack that tries to circumvent the authentication in this paradigm.

In the rest of the chapter, we focus on the challenges an auditor face to search a large amount of snapshots upon receiving a report.

5.2 Effective Search of Snapshots

Once a user reports a harassment incident, an auditor searches the snapshots reported by other users using the same app to find the harasser.

Key challenge. The number of snapshots (for a specific social VR app) reported to Visor's server can be large. For example, assuming an average of 50 active users in a social VR app, there will be about 2 million collected snapshots in one day when capturing a snapshot every 2 seconds (as in Visor). The manual search on such a large number of snapshots is daunting.

In Visor, we attempt at facilitating the search for the auditor. To do so, our goal is to find *promising snapshots* for the auditor to review. The auditor can then review the most promising snapshots until they find the corroborating snapshots.

Key idea. The key idea in Visor is to find snapshots in the same location as the incident. A user, e.g., the victim, reports an incident by submitting a snapshot from their own

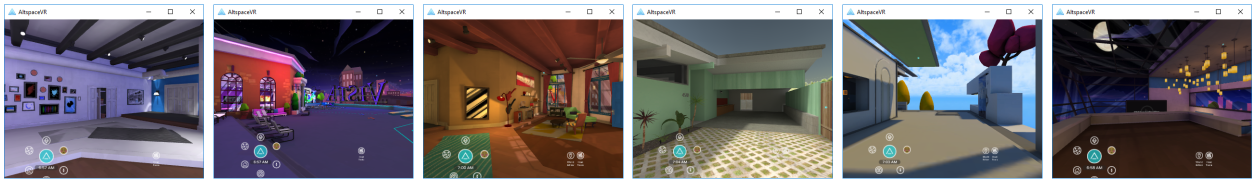

Figure 5.2: Sample locations in AltSpaceVR, demonstrating unique colors and textures.

device. Visor then tries to find promising snapshots from other users by finding those in the same location as the one in the reported snapshot. To achieve this goal in Visor, we (i) perform location-based indexing of the snapshots as they are reported and (ii) perform fast harassment-specific queries on these indices when needed. We next discuss the indexing algorithm and the search algorithm (i.e., the queries).

5.2.1 Location-Based Index

The goal of the location-based index is to infer the location of the user from the reported snapshot, which captures the user's viewpoint. Note that the index does *not* infer the *absolute location* of the user, which is hard to define in a virtual space. Instead, Visor's goal is to infer location proximity between two snapshots (in order to determine which snapshots are from the same location as the reported one).

In Visor, our key idea is to use the visual properties of the snapshot to calculate the index. This is motivated by our observation that virtual spaces have unique and vibrant colors and textures (Figure 5.2). Therefore, if two snapshots capture similar visual properties, they are likely to be in the same location in a virtual space.

Indexing algorithm. We compute the location index by performing an average over RGB values in the pixels of the snapshot. More specifically, the index (I) is a triplet, (R, G, B), where the R, G, and B elements are the average of all R, G, and B values in the pixels, respectively. In this algorithm, we define a distance function between two indices (I_i and I_j) as $dist(I_i, I_j) = |R_i - R_j| + |G_i - G_j| + |B_i - B_j|$. We use this function to define a *closeness* *score* between two indices as $cs(I_i, I_j) = max(L - dist(I_i, I_j), 0)$. The higher the score, the closer two users are (or the lower their distance is). L (which is set to 10 in our prototype) is the max distance we consider in our score, above which the closeness score is 0.

We further improve this algorithm in two ways. First, we perform *tiling*. That is, we divide the snapshots into a configurable number of tiles (i.e., 100 (10×10) tiles in our prototype). Assuming M tiles, each snapshot is an array of M indices ($\overline{I}$), each of which represents the average of RGB values of the pixels in the corresponding tile. The key insight behind this algorithm is to find representative tiles within snapshots and use them for localization. To compute the closeness score between two snapshots, we use the sum of closeness score computed between the indices of all the tiles of two snapshots, or $cs(\overline{I_i}, \overline{I_j}) = \sum_{t_i,t_j} cs(I_{t_i}, I_{t_j})$, where t_i and t_j represent the tiles of snapshots i and j.

Second, we perform *smoothing*. That is, the closeness score of each snapshot also *bleeds into adjacent snapshots*. In other words, the closeness score of each snapshot (with respect to the reported reference snapshot) gets contributions from the closeness score of its adjacent snapshots. We perform this mainly to opportunistically benefit from good representative visual content in adjacent snapshots that might not be present in the current one due to small movements. More specifically, the smoothed closeness score between the reference snapshot S_i and another snapshot S_j is calculated as $smcs(\overline{I_i}, \overline{I_j}) = cs(\overline{I_i}, \overline{I_j}) + \sum_{1 \le n < N} b_n(cs(\overline{I_i}, \overline{I_{j+n}}) + cs(\overline{I_i}, \overline{I_{j-n}}))$, where b_n is the bleeding coefficient with the constraint that $0 < b_n < 1$ and $n < m \rightarrow b_n > b_m$. N determines the number of adjacent snapshots to have any impact at all. In our prototype, we set N so that it covers no more than a 10-second window, hence we set it to 2 when snapshots are 2 seconds apart. We define the bleeding coefficient as a function of how close in time an adjacent snapshot is. More specifically, if the two neighboring snapshots are p seconds apart, we use the coefficient $1/p$.

5.2.2 Querying the Snapshots

Once a harassment report is received, the auditor runs a query on snapshot indices to identify promising snapshots for review. The auditor uses the reported snapshots as a *reference snapshot* and runs a harassment-specific query. We identify two categories of harassment and discuss the query for each.

Category I: Offensive drawings. VR social apps allow users to draw content in the VR world, e.g., 3D drawing on shared spaces and graffiti on the walls. These drawings can then be observed by other users when they are in the same location and, therefore, they have been used for spreading offensive content in the past [33]. The key idea behind the query for this category is to find snapshots in the location of the offensive drawing before the report. More specifically, assuming T_R and $\overline{I_R}$ to be the time and location indices of the reference snapshot (which is reported and includes the drawing in it), the (pseudo) query is: $SELECT\ snapshots\ WHERE\ T < T_R\ ORDER\ BY\ smcs(\overline{I}, \overline{I_R})$, where T and $\overline{I}$ are the database columns capturing the time index and location indices of all snapshots of a particular app. Simply put, this query returns all snapshots before the report time sorted by their closeness score. Snapshots with a high closeness score will have similar content as the reference snapshot and hence tend to be in the location of the drawing. Moreover, since the reference snapshot includes the drawing in it, the query does not favor snapshots from the same location prior to the act of drawing.

Category II: Lewd and offensive actions towards a victim. Examples of this category are offensive gestures, groping, blocking, slapping, and inappropriate touching. The key insight behind this query is that this type of harassment requires the harasser to be close (in location) to the location of the reporter, i.e., the victim, at the time of the incident. More specifically, assuming the harassment to have taken place at time index of T_R and the location index of $\overline{I_R}$ (which are simply the time and location indices of the reference snapshot), the (pseudo) query is: $SELECT\ snapshots\ WHERE\ |T-T_h| < T_{thresh}\ ORDER\ BY\ smcs(\overline{I}, \overline{I_h})$, where T_{thresh} (60 seconds in our evaluation) specifies the time period within which the auditor searches the snapshots. Simply put, this query returns all snapshots within the specified period of time sorted by their closeness score.

Analyzing the query results. Our queries return the snapshots sorted by their closeness score. The auditor needs to manually review the results. To make the manual step easier for the auditor, we create short *automatic slideshows* by merging snapshots from a user that are close in time together, which shows each snapshot for half a second. We then sort the slideshows according to the top 5 high scoring snapshots within them and the overall number of snapshots with scores above a threshold. The auditor can then watch these slideshows until he finds the corroborating snapshot(s).

Chapter 6

Prototype & Data Collection

Prototype. We implement our secure recorder in the KVM hypervisor and use it for the HTC Vive Pro headset [11]. We use SteamVR in a Windows 10 OS to control the headset and run them in a VM on top of the KVM hypervisor. We pass-through the GPU as well as the USB hub used to connect the headset's peripheral devices (such as speakers, microphones, cameras, controllers, and wireless sensors) to the VM. We use an x86-based workstation with 8 CPU cores, a Radeon RX Vega 64 GPU, and a TPM chip v1.2 for the tethered machine for the headset.

A user interested in deploying the secure recorder for their headset needs to take the following steps. We assume that the user already has the VR software, e.g., SteamVR, running in a compatible OS, e.g., Windows 10. The user can continue to this setup when they do not need the secure recorder. To use the secure recorder (and possibly mandatory recording), the user needs to install our KVM hypervisor on the machine, e.g., on a different storage partition. They then boot into this partition and run the Windows partition inside a VM and configure the GPU and headset's passthrough.

We implement the auditing server in C/C++ and run it inside an x86 server with two 18-core

Xeon E5-2697 v4 processors (72 cores, overall, with hyperthreading).

Large-scale data collection. For evaluating the effectiveness of mandatory recording and its search algorithm to find harassers, we need snapshots from a large number of users all using the same social VR app. Since no such dataset is publicly available, we created it ourselves. To be able to scale to a large number of users, we used automation. That is, we developed scripts to launch several instances of the app and move the avatars according to a pre-planned schedule. We deployed up to 99 automated users in this experiment and ran the experiment for 12 hours. Since we did not have access to as many headsets, we used the PC version of the AltSpaceVR app, one of the most popular social VR apps today. We launched and controlled up to 4 users in a workstation. To have about 100 users, we used 25 workstations, most of which were desktops with the following specs: Dell Optiplex 9020 Small Form Factor desktops with Core i5-4590 CPU, Intel integrated GPU, and 8 GB of memory.

A key part of the experiment design was designing the spaces in AltspaceVR for all our users to move in. We created about 125 spaces in total, all as private spaces in order to prevent regular AltSpaceVR users (who are not part of our experiment) from entering any of our controlled spaces and impacting our experiment. Additionally, in order to allow our own avatars to enter the spaces, we friended them with avatars who created the spaces in advance to either get invitations or to follow them to another space (which are needed to enter a private space).

Our emphasis in this experiment was to mimic a social VR setup with real users as much as possible. First, we tried to emulate real user movement within the VR spaces. To create realistic movements, we watched and analyzed existing videos of AltSpaceVR usage uploaded by users on YouTube or Twitch. We found and analyzed about 30 such videos for a total of 14 hours (843 min). All the videos are first-person perspective recordings of actual users playing AltspaceVR for attending an activity or event. By analyzing those videos, we created a distribution of movement functions (i.e., changing directions, moving forward and backward, and transporting within or between spaces), and applied them to automated user movements. Second, we chose the number of users based on reported statistics [2], which showed the maximum number of active users in a social VR app is around 100. Third, we tried to determine the realistic active time of each of our users. [2, 3] show that VR users usually spend between 1 to 2 hours per session on weekdays and longer hours on weekends and show the average active time within AltspaceVR. Also, we observed public events and activities in AltspaceVR take between 1 to 2 hours. Accordingly, we created the distributions for the active time of users. During the experiment, the number of active users varied between 2 and 99.

We automated various actions performed by avatars in this experiment. These activities include expressing emotions through emojis, sending messages, and invitations. We manually performed the rest of the tasks, such as drawing or handling objects, to generate harassment-like incidents for our evaluation (§8). To do this, five people, including some of the authors, attended the lab where the experiment was conducted and performed the manual actions.

During the experiment, we experienced some unexpected difficulties, stemming from bugs in AltSpaceVR. One issue was that some avatars were erroneously shown as offline to other avatars, which prevented them from being automatically transported to another space as offline avatars cannot receive invitations. To address this issue, we manually moved those avatars to other accessible spaces. While we did our best to follow our schedule, we could not completely achieve so due to a large number of users and computers. Another issue was that sometimes an avatar could not see a drawing in the space. Therefore, those cases are naturally excluded when evaluating harassment based on drawings.

Chapter 7

Security Evaluation

Threat Model. We assume that the attacker controls the software running in the tethered machine and controls which hardware components are available in it (e.g., GPU model, amount of RAM, etc.), but that the attacker cannot tamper with individual hardware components or with the VR headset's hardware. We also assume that the attacker cannot read or write to the hypervisor's memory through a physical attack. We note that an attacker who can do so can circumvent Visor's guarantees in various ways including extracting the hypervisor's private key (Pr_H) or altering the snapshots in memory. We note that such an attack is challenging. Yet, it is possible to protect against this powerful attacker through different techniques including keeping the secrets and performing computations on the CPU chip [24] and using trusted execution on the GPU for a secure DMA [36]. We also assume that the attacker cannot leverage side channels to extract secrets from the hypervisor.

To use the secure recorder of Visor, we trust the following components to operate as expected and hence they form the TCB of our solution. First, we trust individual hardware components on the VR device (headset and the tethered machine). Second, we trust the recorder to not be compromised by runtime software attacks. Our emphasis on using a hypervisor with a smaller TCB compared to an OS to implement the recorder is precisely to provide confidence that the recorder is safe against software attacks.

Further, to use mandatory recording, we also trust the following. First, we trust the auditing server including its hardware and software. Second, we trust the social VR app server. A malicious app server can allow malicious users to enter the app against the auditing server's knowledge.

Attackers and security analysis for the secure recorder. To circumvent the secure recorder, the attacker may try to fabricate or tamper with the snapshots. First, they may try to change the snapshots captured by the hypervisor or simply spoof them. This will not be successful as the hypervisor's signature required on the snapshots would not be valid. Second, they may run a malicious version of the hypervisor on their device. This also fails as the attestation message generated using the TPM reveals that the right hypervisor was not running in the device. To defeat this, the attacker might try to falsify the attestation message, e.g., by providing a spoofed certificate for AIK or by spoofing the TPM measurement. These will also fail as an auditor can check the validity of the AIK certificate and the measurement.

Attackers and security analysis for mandatory recording. In this paradigm, in addition to the attacks on the secure recorder discussed above, another form of attack becomes relevant. That is, an attacker may try to hide their virtual identity in the snapshots collected by the auditing server. This way, the system will not be able to effectively identify the attacker, if they are detected to have committed a harassment.

For the analysis, we first consider an attacker who has a single VR headset. This attacker may try to block their VR headset from transmitting any snapshots to the auditing server. In this case, the auditing server invalidates the authentication token and hence the user will be kicked out of the social VR app. Second, the attacker may try to change the virtual identity in the attestation message. However, since the attestation message is signed, this attack will fail.

We now consider an attacker with more than one VR headset. Such an attack, by definition, is more expensive and also harder to execute. Note that, obviously, once a VR headset (i.e., a virtual identity) is identified as having committed harassment and hence punished by the auditing system, the user can use a different VR headset. This is not a successful attack. An attack is successful only if it can prevent any of the headsets from getting identified.

To perform a successful attack, an attacker can take one of the following approaches. First, the attacker may try to use one headset (headset 1) to interact with the auditing server and another (headset 2) to interact with the app server. This way, the auditing server receives benign snapshots from headset 1 while the attacker performs harassment using headset 2. To do this, they can get the authentication token using headset 1 and pass it to headset 2 to use to enter the social VR app. This approach will fail as the app server will not be able to verify the virtual identity of headset 2 as the token is for the virtual identity of headset 1. Second, to defeat this, the attacker can reuse the attestation message from one of the headsets for the other one. This will fail since, as explained in §5.1, the auditing server and the app server each sends a nonce value to the headset to be included in the attestation message. Acting as a challenge, this nonce value prevents reuse of the attestation message. Third, to defeat this, an attacker may try to forward the attestation challenge (i.e., the nonce) from headset 2 to headset 1, generate the response in headset 1, send it back to headset 2, and then forward the response to the app server. This attack can be defeated in several ways. For example, the app server can monitor the response latency, similar to [21], or it can extract the challenge directly from the message delivered by the network interface and include the message preamble (which includes the sender address) in its signed response. Finally, the attacker may try to perform the identity verification with the app server on headset 1 (which is also used to receive the token from the auditing server), and then live-migrate the VR software to headset 2. This attack can be easily defeated by the app server performing the virtual identity verification periodically.

Chapter 8

Evaluations

In this chapter, we evaluate the overhead and effectiveness of Visor.

8.1 Secure Recorder

Visor's secure recorder causes performance overhead (to the VR app) for two reasons: first, it requires the VR software to run within a VM. Second, it intercepts select interactions of the GPU driver in the untrusted OS with the GPU hardware. To quantify the overhead, we measure the Frames Per Second (FPS) achieved by the app and overall system CPU utilization for three scenarios: Bare-metal (when the VR software, the app, and the Windows 10 OS run directly on the physical machine), VM (when they run within a VM but the recorder is integrated in the hypervisor), idle recorder (when the user has enabled the secure recorder but is not using it to take snapshots, and active recorder (when the user is actively taking snapshots using the recorder). For the last one, we consider the user takes snapshots at various frequencies: F1 (1 snapshot per second), F2 (1 snapshot per 2 seconds), and F3 (1 snapshot per 5 seconds).

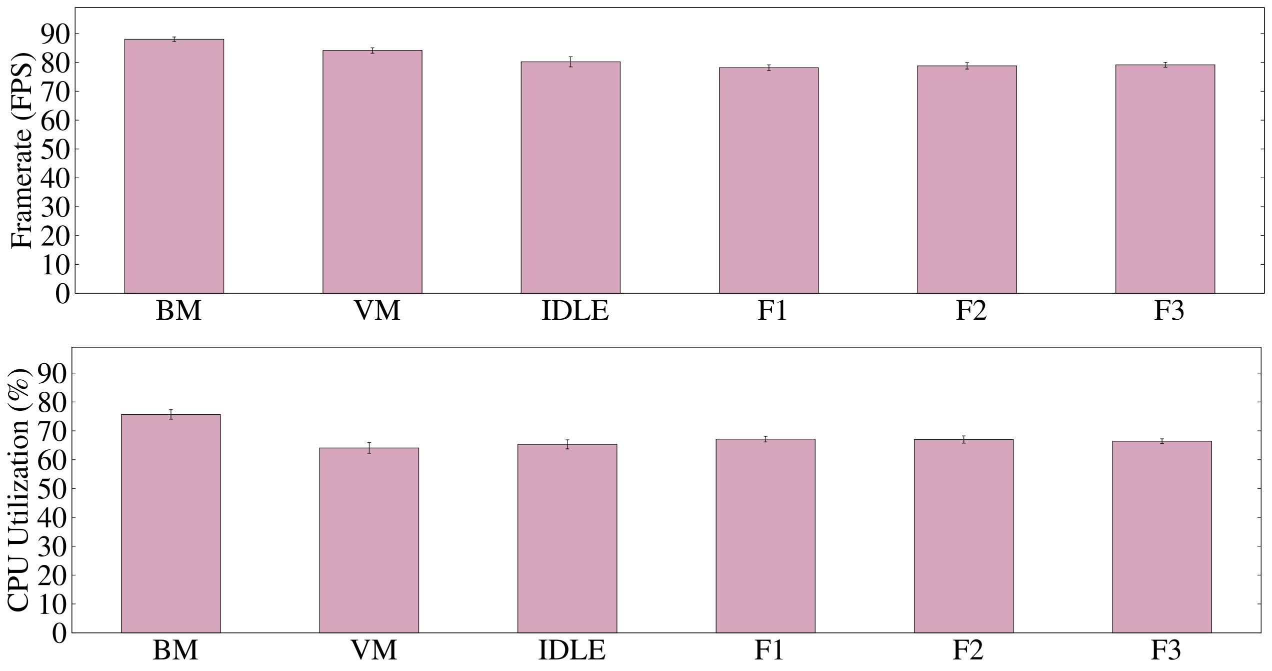

Figure 8.1: (Top) VR app framerate and (Bottom) system CPU utilization when running the VR platform bare-metal in a physical machine (BM), in a VM, and when integrating Visor's secure recorder but not taking any snapshots (IDLE) and when actively taking snapshots with different frequencies (F1, F2, and F3).

Figure 8.1 shows the results. Each histogram shows the average and standard deviation of three runs. The figure shows that using the VM adds 4% overhead to the framerate. Moreover, the secure recorder (when not capturing snapshots) adds about another 6% overhead to the framerate and increases the CPU utilization by about 2%. Finally, when actively taking snapshots, the secure recorder adds about another 2% overhead to the framerate and increases the CPU utilization by about 4%. Overall, the recorder's overhead is small and does not impact the user experience significantly as the headset achieves a high framerate.

Snapshot collection. In addition, we measure the time it takes for the secure recorder to take one snapshot. That is, we measure the time from when a program in the VM asks the secure recorder for a snapshot until when the snapshot is provided to the application. Our measurements show that this time is about 112 ms.

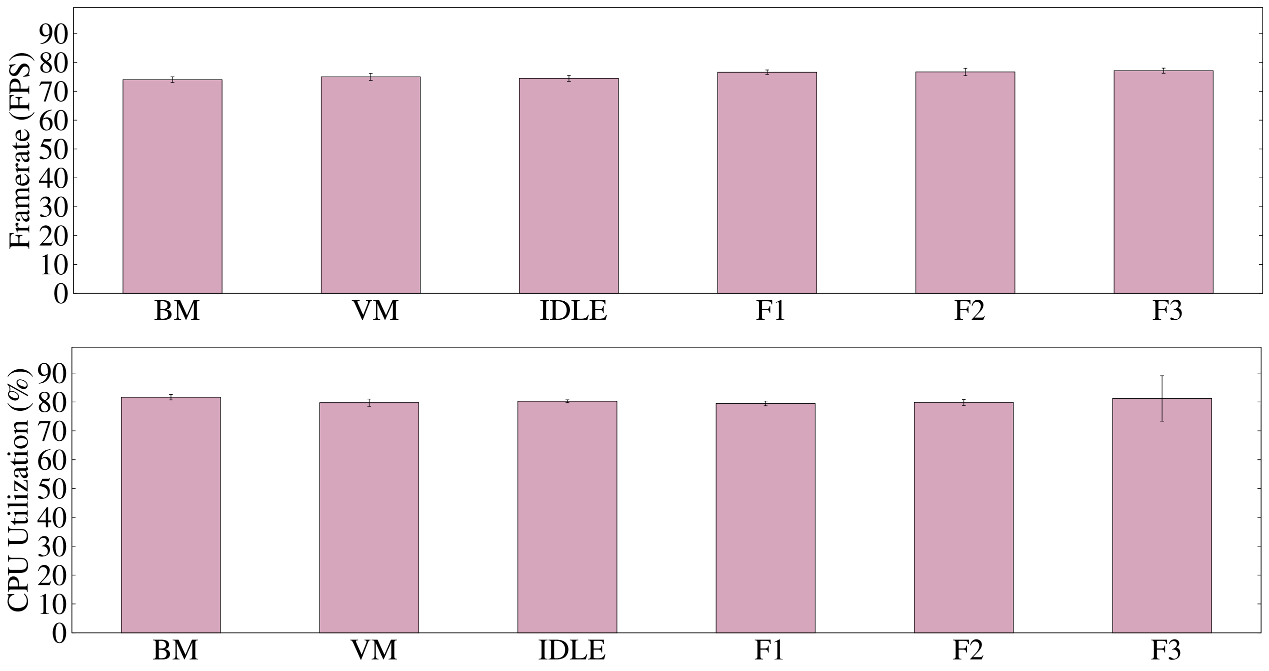

Figure 8.2: (Top) VR app framerate and (Bottom) system CPU utilization when using the mandatory recording paradigm with different snapshot resolutions and frequencies.

8.2 Mandatory Recording

We measure the overhead of mandatory recording. In this paradigm, in addition to capturing snapshots, the snapshots need to be transmitted to the auditing server. To minimize the impact on network bandwidth, we downsample the snapshots (which have the resolution of 2880×1600). In the experiments, we try to empirically support the choice of transmitted snapshot resolution and the frequency of capturing them. To do so, for most experiments, we report the results for some combinations of three resolutions and three transmission frequencies. The resolutions are R1 (960×550), R2 (600×300), and R3 (450×225). The frequencies are the same as the ones used in the previous experiments. Visor's choice is R2 and F2, which we highlight in the figures. We choose these parameters as they are effective in our search experiments and have a smaller performance and resource overhead compared to some other parameters with comparable effectiveness.

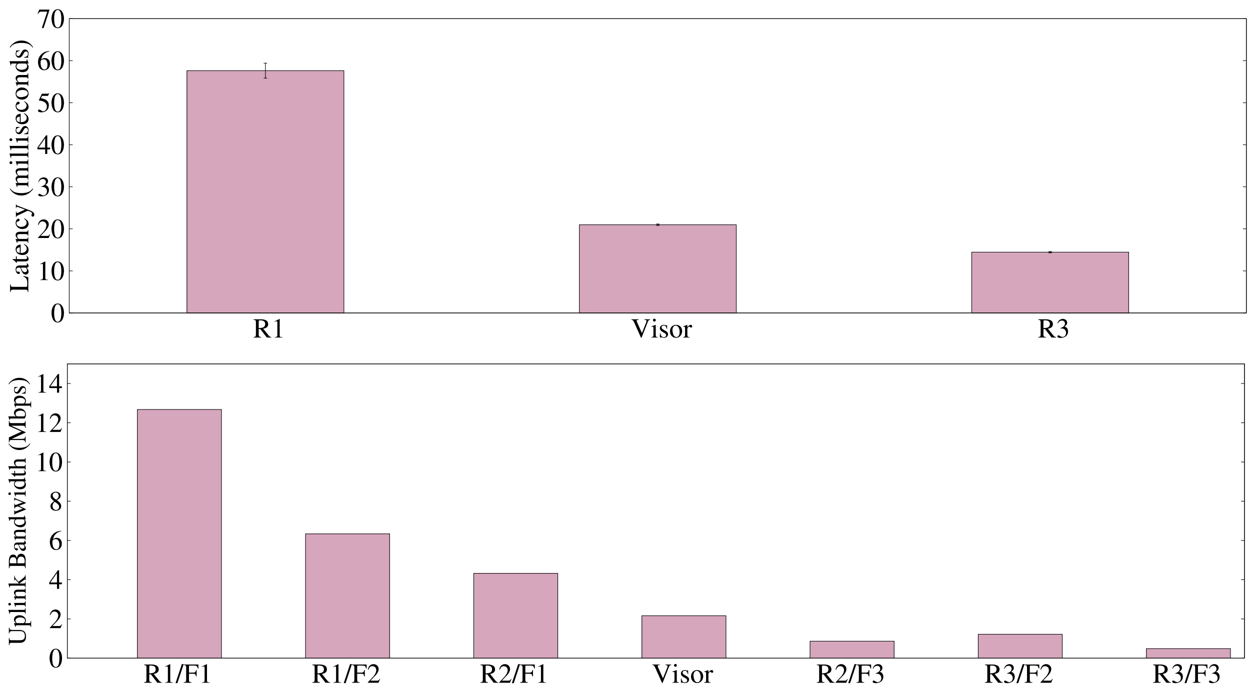

Figure 8.3: (Top) Latency of collecting a snapshot from a VR headset. (Bottom) Uplink bandwidth required for collecting snapshots.

Figure 8.2 shows the results. The figure shows that Visor adds about another 9% overhead to the framerate (compared to VM) and increases the CPU utilization by about 24%. Compared to just using the secure recorder, the overhead for mandatory recording is higher. But overall, the overhead is still small allowing the headset to achieve a high framerate.

Remote snapshot collection. We measure the time it takes for the auditing server to collect one snapshot from a VR headset. This includes the time for the server to send an initiating message to the headset and the time for the headset to capture a snapshot using GPU DMA command side-loading, to prepare the snapshot, to digitally sign it, and to send it to the server. Figure 8.3 (Top) shows the results (again averaged over 3 runs). It shows that the auditing server is able to collect a sample in 21 ms, which is much smaller than the 2-second window, in which a snapshot needs to be captured.

Bandwidth requirement. We report the network uplink bandwidth required to transmit the snapshots captured by the recorder to the auditing server. We show the results in

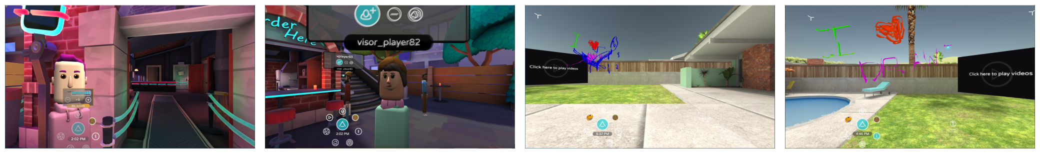

Figure 8.4: Sample snapshots from the incidents in our experiments. (Leftmost two) An action incident. (Rightmost two) A drawing incident. (First, Third) Snapshots reported by victims. (Second, Fourth) Snapshots from harasser's view corroborating the report.

Figure 8.3 (Right). The results show that the required bandwidth highly depends on the snapshot resolution and capture frequency. For the configuration used for Visor, the required bandwidth is 2.16 Mbps. Given that today's wired and wireless connections provide tens to hundreds of Mbps or even Gbps of bandwidth, the bandwidth requirement of mandatory recording is modest.

Latency of TPM-based remote attestation. We measure the time it takes for either the auditing server or the app server to perform remote attestation. This includes the time for the server to send a nonce to the headset, the time for the headset to capture a TPM measurement, to generate a new key pair, to construct the attestation message, and to send it to the server. It also includes the time for the server to process and verify the message. Our measurements with 20 runs of the attestation handshake show that remote attestation takes 0.8 seconds on average, with standard deviation of 0.01 seconds. This time is small and will not impact the user experience especially given that the attestation is performed a few times.

Effectiveness of indexing and search algorithms. To evaluate the effectiveness of these algorithms, we choose 10 *incidents* in our large-scale experiment dataset (§6). Five incidents are drawings in virtual spaces and the other five incidents are actions performed by an avatar towards another one. For each incident, we use a snapshot that is reported by a victim as the reference snapshot. Then, we execute the corresponding query using the provided reference (§5.2.2). Figure 8.4 shows sample reference snapshots and corroborating snapshots from the harassers' device (for which the auditor looks) for two of the incidents in our evaluation.

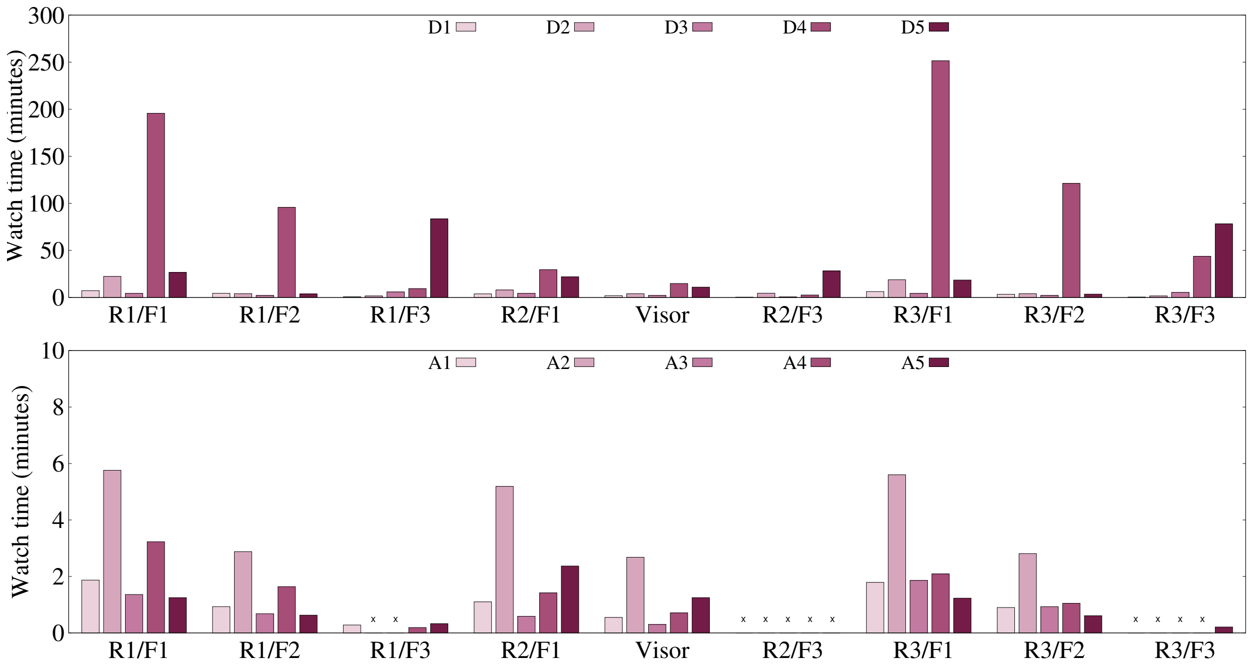

Figure 8.5: The manual effort, in time spent watching snapshot slideshows, by the auditor to find a harasser for (Top) drawing incidents and (Bottom) action incidents. × highlights cases that no corroborating snapshots were found.

As mentioned in §5.2.2, we merge the snapshots into slideshows for the auditor to watch (one snapshot per 0.5 seconds). Figure 8.5 shows how many minutes of slideshow the auditor needs to watch before they find corroborating snapshots. The results show that the auditor finds the harasser in 6.8 minutes on average for drawing incidents and in 1.1 minutes for action incidents (and 4 minutes on average when combined) with Visor. The results from using other snapshots resolutions and frequencies are insightful. First, increasing the resolution (R1) does not improve the results. Second, increasing the frequency (F1), surprisingly, results in longer watch times mainly because it finds many more similar snapshots for the auditor to review. Third, lowering the resolution (R3) also achieves worse results mainly because the location index does more poorly. Finally, lowering the frequency (F3) has poor results, especially for action incidents, mostly because no snapshots may capture the incident, which might not take more than a few seconds.

To further demonstrate the effectiveness of the location index, we also calculate the average time needed to find the harasser if the auditor would randomly view the snapshots. In

Incident	Average score of negatives	Average score of positives	Average score of negatives watched by auditor	Average score of positives watched by auditor
D1	97.16	2490.15	1329.68	4128.10
D2	561.18	6517.72	5354.53	9396.55
D3	194.26	6027.64	3520.44	7744.38
D4	242.84	1712.15	2121.68	2440.45
D5	1214.11	4700.18	11642.41	9054.92
A1	79.58	1204.35	574.0	1344.71
A2	467.15	769.24	1219.73	948.35
A3	109.5	420.7	798.53	612.29
A4	270.24	2867.38	1452.24	614.98
A5	357.04	1076.37	1689.79	1270.84

Table 8.1: The effectiveness of the location index.

this case, we assume the auditor can still query the time of the snapshot, but not location. We assume similar parameters as Visor: that is, the VR headsets report 1 snapshot per 2 seconds and the auditor spends 0.5 seconds to view each snapshot. In this case, the auditor needs to spend 2275.1 minutes on average for the drawing incidents and 4.6 minutes for the action incident, an increase of 33,525% and 321.5%, respectively. The benefit the location index is much higher for drawing incidents since for action incidents, the time query allows the auditor to focus on snapshots from the period of time of the incident. For the drawing, however, the auditor does not know the time the drawing was performed.

While these results demonstrate the overall effectiveness of the search algorithm, they do not reveal the accuracy of the location index. Therefore, for the same queries, we report the average closeness score computed for positive and negative snapshots (i.e., snapshots that are and are not actually in the location searched by the query) among all the snapshots. We also report the average scores among the snapshots viewed by the auditor, which are the ones with scores equal to or higher than the corroborating ones the auditor is looking for. To do this, we manually inspected all the snapshots in the dataset and identified the ground truth for the location, a process that took us about 5 days. Table 8.1 shows the results. As can be seen, the positives mostly receive better scores than negatives.

Storage requirements in auditing server. The auditing server needs to collect many snapshots. Assuming 50 users on average active in a social VR app, the auditing server uses 7.8 TB of storage space for snapshots collected per week, which fits in a single hard disk drive. Given this, we believe it is cheap and feasible for the server to keep snapshots of several months of activity at a time.

Chapter 9

Conclusion

We introduced Visor, a system solution for providing trustworthy and effective reports for social VR. It comprises two components: a secure recorder, which is deployed on the VR headset and captures tamper-proof and irrefutable snapshots of displayed content; and mandatory recording, a novel paradigm that collects snapshots from all the users of the app and helps an auditor search them to find harassers. We presented novel solutions to provide strong security guarantees for Visor and to make the search of the snapshots more efficient. Our evaluation with a prototype on HTC Vive Pro and a large snapshot dataset collected from 99 automated users in a 12-hour experiment showed that Visor is secure, does not incur significant overhead, and enables the auditor to find a harasser with minutes of manual effort.

Bibliography

[1] AltspaceVR redesigns user experience. https://altvr.com/new-host-tools/, 2016.

[2] An ongoing analysis of Steam's concurrent players: AltspaceVR. https://steamcharts.com/app/407060, 2016.

[3] POLL - How many hours a day do you play VR? https://www.reddit.com/r/oculus/comments/9sqr7y/poll_how_many_hours_a_day_do_you_play_vr/, 2018.

[4] Virtual Reality Device Market Declines in 2018 But Outlook Remains Positive. https://www.ccsinsight.com/press/company-news/3726-virtual-reality-device-market-declines-in-2018-but-outlook-remains-positive/, 2018.

[5] Comfort and Safety. https://recroom.com/comfortandsafety, 2019.

[6] Recommended Computer Specs (for Vive headsets). https://www.vive.com/us/ready/, 2019.

[7] Recroom: Report system abuse. https://www.reddit.com/r/RecRoom/comments/8s5vb9/report_system_abuse/, 2019.

[8] VRChat Safety and Trust System. https://docs.vrchat.com/docs/vrchat-safety-and-trust-system, 2019.

[9] Windows Defender System Guard: How a hardware-based root of trust helps protect Windows 10. https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-system-guard/how-hardware-based-root-of-trust-helps-protect-windows, 2019.

[10] Altspacevr community standards. https://help.altvr.com/hc/en-us/articles/115003528793-AltspaceVR-Community-Standards, 2020.

[11] Htc vive pro. https://www.vive.com/us/product/vive-pro/, 2020.

[12] Oculus abuse report. https://support.oculus.com/1126372434098455/?locale=en_US, 2020.

[13] Oculus abuse report web form. https://support.oculus.com/report-user/, 2020.

[14] Oculus: Appeal the code of conduct violation. https://support.oculus.com/885981024820727/, 2020.

[15] Oculus code of conduct. https://support.oculus.com/1694069410806625/, 2020.

[16] Recroom code of conduct. https://recroom.com/code-of-conduct, 2020.

[17] Recroom report system. https://recroom.com/reporting, 2020.

[18] Vrchat guideline on filing useful report. https://help.vrchat.com/kb/article/26-i-want-to-report-someone/, 2020.

[19] Vrchat report system. https://help.vrchat.com/new, 2020.

[20] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert. Intel Virtualization Technology for Directed I/O. *Intel Technology Journal*, 2006.

[21] B. Aziz and G. Hamilton. Detecting man-in-the-middle attacks by precise timing. In *Proc. IEEE Int. Conf Emerging Security Information, Systems and Technologies*, 2009.

[22] J. Belamire. My First Virtual Reality Groping. https://medium.com/athena-talks/my-first-virtual-reality-sexual-assault-2330410b62ee#.8lcy2o2bh, 2018.

[23] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. A. Yassour. The Turtles Project: Design and Implementation of Nested Virtualization. In *Proc. USENIX OSDI*, 2010.

[24] P. Colp, J. Zhang, J. Gleeson, S. Suneja, E. De Lara, H. Raj, S. Saroiu, and A. Wolman. Protecting Data on Smartphones and Tablets from Memory Attacks. In *Proc. ACM ASPLOS*, 2015.

[25] M. Ehrenkranz. Oculus Just Rolled Out Updated Anti-Harassment Tools. https://gizmodo.com/oculus-just-rolled-out-updated-anti-harassment-tools-1829976156, 2018.

[26] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, D. Tsafrir, and A. Schuster. ELI: Bare-Metal Performance for I/O Virtualization. In *Proc. ACM ASPLOS*, 2012.

[27] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *Proc. USENIX ATC*, 2006.

[28] M. Liu, T. Li, N. Jia, A. Currid, and V. Troy. Understanding the Virtualization "Tax" of Scale-out Pass-Through GPUs in GaaS Clouds: An Empirical Study. In *Proc. IEEE HPCA*, 2015.

[29] S. Liu. Consumer spending on augmented and virtual reality (AR/VR) content and apps worldwide from 2016 to 2021 (in million U.S. dollars). https://www.statista.com/statistics/828467/world-ar-vr-consumer-spending-content-apps/, 2018.

[30] S. Mirzamohammadi and A. Amiri Sani. The Case for a Virtualization-Based Trusted Execution Environment in Mobile Devices. In *Proc. ACM Asia-Pacific Workshop on Systems (APSys)*, 2018.

[31] J. Outlaw. Harassment in Social VR: Stories from Survey Respondents. `https://medium.com/@jessica.outlaw/harassment-in-social-vr-stories-from-survey-respondents-59c9cde7ac02`, 2018.

[32] J. Outlaw. VIRTUAL HARASSMENT: THE SOCIAL EXPERIENCE OF 600+ REGULAR VIRTUAL REALITY USERS. `https://drive.google.com/file/d/1afFQJN6QAwmeZdGcRj9R4ohVr0oZNO4a/view`, 2018.

[33] J. Outlaw and B. Duckles. WHY WOMEN DON'T LIKE SOCIAL VIRTUAL REALITY: A STUDY OF SAFETY, USABILITY, AND SELF-EXPRESSION IN SOCIAL VR. `https://static1.squarespace.com/static/56e315ede321404618e90757/t/59e4cb58fe54ef13dae231a1/1508166522784/The+Extended+Mind_Why+Women+Don%27t+Like+Social+VR_Oct+16+2017.pdf`, 2017.

[34] A. Stanton. Dealing With Harassment in VR. `https://uploadvr.com/dealing-with-harassment-in-vr/`, 2016.

[35] M. Thibault. VR Social Networks will be all the rage in 2018. `https://medium.com/@tickarawr/vr-social-networks-will-be-all-the-rage-in-2018-fda5093c65`, 2018.

[36] S. Volos, K. Vaswani, and R. Bruno. Graviton: Trusted Execution Environments on GPUs. In *Proc. USENIX OSDI*, 2018.

[37] D. Wilson. Triple Buffering: Why We Love It. `https://www.anandtech.com/show/2794/2`, 2009.