Enhancing the Security of VR Applications By Detecting Immersive and Data-Integrity Threats

Bhargav M V
Computer Science and Engineering
PES University
Bangalore, India.
bhargav2650@gmail.com

B G Sourav
Computer Science and Engineering
PES University
Bangalore, India.
bgsouravbg@gmail.com

Kamal Sab
Computer Science and Engineering
PES University
Bangalore, India
sabkamal61@gmail.com

K J Subodh
Computer Science and Engineering
PES University
Bangalore, India.
kjsubodh@gmail.com

Dr. Nagasundari S
Computer Science and Engineering
PES University
Bangalore, India.
nagasundaris@pes.edu

***Abstract*— This Paper presents the results of research done on vulnerabilities in the VR applications. VR applications are susceptible to attacks that compromise data integrity and immersion, such as buffer-overflow exploit, ad-fraud, chaperone and disorientation attacks. In response, we propose building a VR application that can facilitate remote communication and collaboration while detecting and mitigating such attacks. Our solution involves comprehensive testing to identify potential attack vectors and develop effective mitigation techniques. The proposed VR application will be designed to provide an immersive experience while ensuring the security and integrity of the data exchanged within it. This paper presents the methods developed to simulate immersive and data integrity attacks, such as chaperone, disorientation, buffer overflow exploit, and ad-fraud attacks on the proposed VR application. This paper also presents mitigation methods developed and integrated into the proposed VR application to mitigate the impact of the above mentioned attacks. By doing so, we aim to enhance the security of VR applications and increase their usefulness for various remote collaboration scenarios. Through this research paper, we aim to contribute to the growing field of VR security and encourage further research in this area.**

***Keywords*— Virtual Reality, Virtual Meeting, WebXR, Web 3.0, Unity3D, Security, Virtual Reality Vulnerabilities.**

I. Introduction

Augmented and Virtual reality are regarded as two of the most world-changing technologies of the 21st century. For a good user experience we need a Head Mounted Display with a binocular view such that each eye has a slightly different viewing angle. These two streams of videos or images combine to form an overlap, giving the viewer immersive and interactive experiences with the virtual world. Virtual conferencing applications can offer more than an alternative to conventional face-to-face events. Scientific collaborations that emerge from conference meetings can be more novel, cross-disciplinary and more frequently cited than projects between two researchers in the same institution [1]. We contend that players immersed in VR can be manipulated, leading to exploitation. Although research dedicated to security and privacy of Mixed Reality systems have been conducted[2], we found no applied work regarding the vulnerabilities presented by room-scale immersive VR experiences. Therefore, we explored some novel attack vectors in VR. In this paper, we discuss more about the specific tools which can be used to make such an application. For the purpose of keeping our research focused, we explore the vulnerabilities in VR applications made using Unity. We shall also discuss a few attacks that are specific to VR applications. Since there are many cyber attacks such as immersive attacks, data integrity attacks, hardware attacks and many others that a VR application is vulnerable to, we shall restrict our discussion to only immersive and data integrity attacks.

We also discuss mitigation techniques that help us in enhancing the security of VR applications. The rest of the paper is organized as follows. The first two sections introduce and share information about background and related work in the field of Virtual reality. In the third section we discuss the implementation details of a Virtual conference application. The next section is spent on discussing the different attacks and exploits that can be performed in a VR application like the virtual conferencing application. The next section presents a few mitigation techniques for threats discussed in the previous section. Finally, we conclude the paper and briefly touch on a few possible directions for future work.

II. Background

The popularity of VR based applications have rapidly increased in recent years. However, despite the attention, investment, and the promise of Metaverse, there are various security related issues and concerns that have to be touched upon before the Metaverse can truly be adopted in practice for serious applications. The realization of the Metaverse requires the use of visualization technologies such as Virtual Reality (VR) and Augmented Reality (AR). This aspect of the VR applications is suspected to give rise to emerging cybersecurity threats that have not been studied much yet. As such, the purpose of this research work is to conduct studies regarding the cybersecurity threats faced by VR applications.

As VR is a multidisciplinary topic, the intention of this section is to provide a background of the field, and to discuss the work that is already done in this field of research.

In Spite of the growth in popularity of WebXR, many applications developed using WebXR are insufficiently

tested and are delivered with unexpected bugs, which lead to exploits. For example, it was found that there were hundreds of open issues waiting to be resolved in the GitHub repository of A-Frame [6]. These bugs significantly degrade user experience and can potentially cause undesirable consequences. For the study in this paper[3], 368 real bugs were discovered and gathered by analyzing the issue reports and release notes of 33 WebXR projects hosted on GitHub. 320 of the studied bugs were classified into functional issues and performance issues.
Functional issues were the most common sort of WebXR bugs, with 162 out of 247 such bugs being application specific. These bugs are caused due to the design of the application rather than the Rendering Engine issues. It was also found that 85 of the 247 functional issues are related to scene rendering. These rendering issues can be divided into: (1) mis-rendering of objects and (2) absence of objects. Figure 3 gives an example of mis-rendered objects.

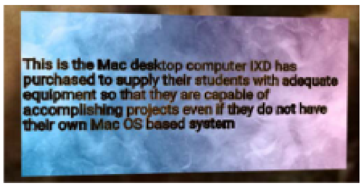

This is the Mac desktop computer IXD has purchased to supply their students with adequate equipment so that they are capable of accomplishing projects even if they do not have their own Mac OS based system

Fig. 1: An example of mis-rendering of objects(blurred text)

Besides functional issues, performance issues are also common in WebXR projects. In total, 36 out of the total bugs discovered were observed to be performance issues. They could be classified into 4 subtypes, such as high memory consumption, high CPU utilization, abnormal crashes and dipped frame rate. The study of this paper[3] highlights the need for research in the area of cybersecurity in VR applications, as the development, nature of bugs, events and interactions are different compared to that of traditional javascript applications.

III. Design and Implementation of Virtual Conference Application

A virtual conferencing application is a virtual reality application that helps remote teams, people separated by geographical distance to collaborate with each other in an immersive Virtual experience. Such an application would include features like Spatial Audio, Hand Gestures, Multi-platform support , etc. A user of this application can host a meeting room or join a room hosted by another user. On joining a meeting room, users are given unique avatars to identify themselves and other users in the meeting. To provide a much more immersive experience, the application includes interactions between the user and the VR environment, such as writing on a board, uploading text or a 3D model into the VR room, and the voice communication between the users in the room etc.
The VR application is developed using two main tools, Unity and Blender. Unity is a platform that lets developers create 3D games, VR applications and many other applications. Blender is a 3D modeling software. We design all the 3D models that are required in our application using Blender, and then import the models into the Unity editor to model the interactions and create the rest of the application.
A conference room includes a board and monitor, because the users need to be able to import the text on the monitor and also be able to write on the board so that it gives a complete real time meeting experience. Below is the figure that simulates the conference application.

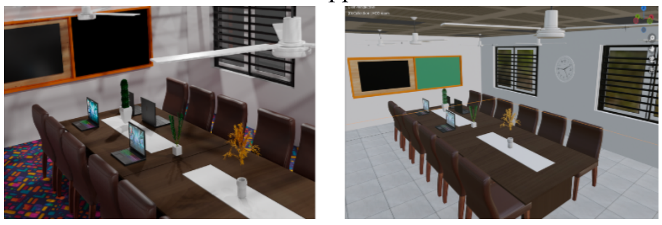


Fig. 2: Conference room application Figures

A multiplayer functionality is a necessity in the application. We use Photon PUN(Photon Unity Networking), a popular framework for Unity that provides a high-level API for implementing multiplayer functionality. PUN is easy to use and supports both real-time and turn-based games. We utilize Unity's XR Interaction toolkit to model the interactions between the user and the environment. We also use packages such as unity standalone file browser and OBJ importer to model the interactions. This helps us design high-level, component- based, interaction systems in unity.
To avoid and mitigate the security risks due to the file upload features, dynamic algorithms for risk mitigation and behavioral analysis of the users based on analysis of the file uploaded would be integrated into the application. The application would also act as a platform for advertising by providing an immersive virtual experience to draw in customers and potential buyers. Advertisements in immersive virtual experience work very differently compared to advertisements in web browsers and are thus vulnerable to exploits similar to clickjacking[9]. The application would provide a safe platform for 3rd party ad service providers to safely advertise without exploitation of the users. VR applications are also vulnerable to immersive threats that compromise the security of the users of the VR application, by making users in the VR environment feel uncomfortable, or lead to harm to the user in the real world, when users are immersed in the virtual world. The application would safeguard users from such attacks, thereby providing a safe and secure immersive virtual experience to the users.

IV. Threat Simulation

A. Chaperone Attack

The chaperone system is used to display the boundaries of Virtual Reality(VR) interaction areas. The chaperone_info.vrchap is a json file that stores all the collision bounds which decide the dimensions of the virtual reality play area and all the coordinates are in meters. The Chaperone system is a safety feature that needs to be considered before developing any VR application. The primary purpose of the chaperone is to enhance the users safety and prevent accidents or injuries. The headset considers the realtime area and virtual environment and defines the chaperone system. Steam acts as a communication channel between the VR application and the VR device. When an application is started for the first time and connected to steam, steam requests the user for the safe play area in the real world that the user can engage in without bumping into physical objects in the real world. These details are stored in the chaperone_info.vrchap file.

Assuming that the file gets manipulated, and there is a change in the values of the chaperone boundaries of the chaperone system, this may lead to the user bumping into real world objects or slipping on the floor of the real world leading to accidents.

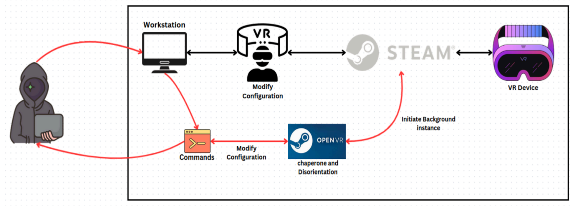


Fig.3: Immersive Exploit

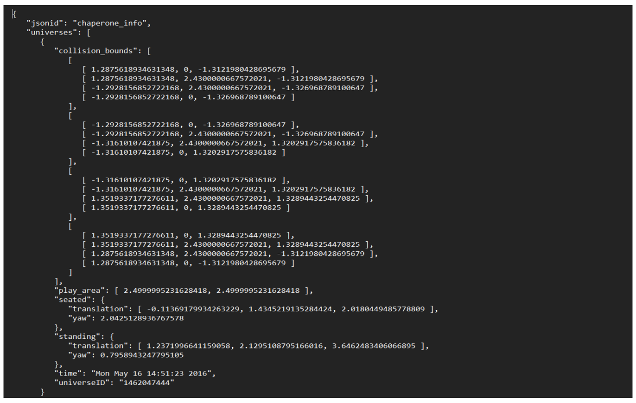
The chaperone file can be manipulated through the openVR library[8]. We utilize the openVR library from which we can run the application from the background. We can manipulate the chaperone_info.vrchap file using a python script, using the following commands:

- vr::VRChaperoneSetup()->ReloadFromDisk(vr:: EChaperoneConfigFile_Live);
- vr::VRChaperoneSetup()->CommitWorkingCopy (vr::EChaperoneConfigFile_Live);

The execution of these functions do not affect the rendering of the VR environment and the changes are reflected when the VR environment is reloaded, due to which the user in the VR environment remains unaware of the manipulation.

B. Disorientation Exploit

Disorientation exploit and the Chaperone exploit share a common strategy of manipulating the VR user's position and rotation within the virtual environment. This manipulation aims to cause Visually Induced Motion Sickness (VIMS) by presenting visual cues that don't align with the user's actual physical movements.
To simulate this, we made adjustments to the translation and yaw values of chaperone_info.vrchap, which is the linear and radial movements respectively. A python script would modify the values and a chaperone file reload would suffice to cause the intended disorientation in the VR environment.
This exploit, after the chaperone modification, disrupts the user's VR experience by creating disparities between their actual movements and those replicated within the virtual environment. Radial movement becomes unpredictable, either excessively rapid or sluggish, while linear translation within the VR space also becomes irregular. Such inconsistencies between real-world actions and virtual responses can lead to disorientation among users immersed in the VR environment.



```
{
   "jsonid": "chaperone_info",
   "universes": [
      {
         "collision_bounds": [
            [
               [ 1.2875618934631348, 0, -1.3121980428695679 ],
               [ 1.2875618934631348, 2.4300000667572021, -1.3121980428695679 ],
               [ -1.2928156852722168, 2.4300000667572021, -1.326968789100647 ],
               [ -1.2928156852722168, 0, -1.326968789100647 ]
            ],
            [
               [ -1.2928156852722168, 0, -1.326968789100647 ],
               [ -1.2928156852722168, 2.4300000667572021, -1.326968789100647 ],
               [ -1.31610107421875, 2.4300000667572021, 1.3202917575836182 ],
               [ -1.31610107421875, 0, 1.3202917575836182 ]
            ],
            [
               [ -1.31610107421875, 0, 1.3202917575836182 ],
               [ -1.31610107421875, 2.4300000667572021, 1.3202917575836182 ],
               [ 1.3519337177276611, 2.4300000667572021, 1.3289443254470825 ],
               [ 1.3519337177276611, 0, 1.3289443254470825 ]
            ],
            [
               [ 1.3519337177276611, 0, 1.3289443254470825 ],
               [ 1.3519337177276611, 2.4300000667572021, 1.3289443254470825 ],
               [ 1.2875618934631348, 2.4300000667572021, -1.3121980428695679 ],
               [ 1.2875618934631348, 0, -1.3121980428695679 ]
            ]
         ],
         "play_area": [ 2.4999995231628418, 2.4999995231628418 ],
         "seated": {
            "translation": [ -0.11369179934263229, 1.4345219135284424, 2.0180449485778809 ],
            "yaw": 2.0425128936767578
         },
         "standing": {
            "translation": [ 1.2371996641159058, 2.1295108795166016, 3.6462483406066895 ],
            "yaw": 0.7958943247795105
         },
         "time": "Mon May 16 14:51:23 2016",
         "universeID": "1462047444"
      }
```

Fig. 4: chaperone_info.vrchap file

C. Buffer Overflow exploit

Implementation of buffer overflow attack involved testing of different data structures in unity after building the .NET executables. On analysis, we found the limited allocation of memory for the textMeshPro text attribute vulnerable to overflow exploits.
We have a specific feature in the virtual conference application, which gives the users the control to upload and display text files on the virtual TV, using the TextMeshPro asset. This feature enables the users to upload a file of any size, which also involves files that are carefully crafted to get access to some functionality of the server. This of course is similar to the vulnerability present in the "gets" function in c. Through some careful analysis, we have discovered that a game object created using the textMeshPro asset has a text attribute, the length of which is bounded. This is because of the fact that the textMeshPro renders the entirety of the text on screen with very little loss of clarity, and thus can only accommodate a limited amount of text on screen and thus, allocates limited memory to store the text, which is stored in the stack memory. Thus when we upload a text file of a higher length, it causes a stack buffer overflow leading to potential exploits.
Heap buffer overflow was simulated after considering the various issues on the internet regarding the upload of a 3D object, which causes out of memory exceptions. Using the memory profiler in Unity, one can observe that the creation of new objects is stored in the heap memory. This varies across different machines, depending on the RAM and GPU memory buffers available. We import the native standalone file browser package, to upload text files as well as 3D objects as obj files at runtime(during the runtime of the game instance) in unity. We have implemented the feature to upload 3d objects as obj files, using the fast obj importer. And after some rudimentary tests, on uploading the 300 Mb blender file available on the internet, our machine crashed due to lack of memory. This varies from machine to machine depending on the RAM and GPU capacities. In a multiplayer environment, every game object in the meeting is instantiated in every user's instance, one can easily buy a high end machine, and upload a huge 3D file, causing the instances of every other user who uses a low end machine to crash and leave the meeting.

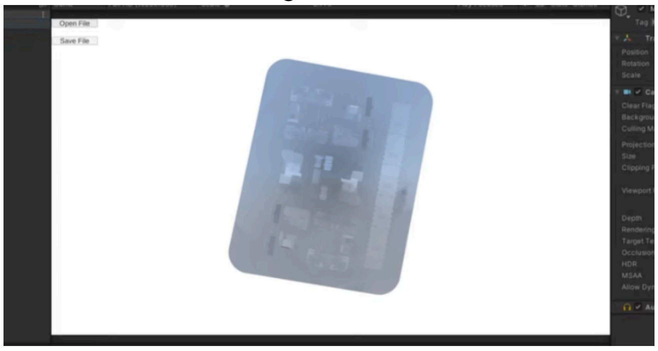
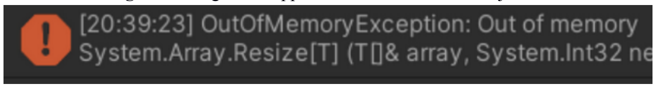


Fig 5: working of the application for a normal 3d obj file.



```
[20:39:23] OutOfMemoryException: Out of memory
System.Array.Resize[T] (T[]& array, System.Int32 ne
```

Fig 6: the exception observed when a very large file is uploaded by the user and the application crashes.

D. *Ad-Fraud*

Ad fraud encompasses various deceptive practices like click fraud, impression fraud, and attribution fraud. Tactics such as ad stacking and malicious redirects further exacerbate the problem, resulting in financial losses and eroding trust in the advertising ecosystem. For the purpose of this paper, we are focusing on unauthorized scaling and URL misdirection as the main ad-fraud techniques.

This method of exploitation involves the manipulation of ad dimensions beyond authorized parameters, often achieved through the direct alteration of scale properties via code manipulation within Unity applications. By exploiting vulnerabilities in the ad display process, fraudsters seek to exaggerate the visibility or impact of advertisements, thereby misleading advertisers and deceiving users. In this research paper, our focus is directed towards examining the forced scaling through code method as a prominent manifestation of ad fraud within the Unity development environment. We achieved this by acting as the ad network providers, and taking a fraudulent script as input to be attached to the ad object. As soon as the script is attached the object starts scaling haphazardly providing a displeasuring experience to the people in the meeting.

In URL misdirection we are dealing with the scenario where the fraudster provides fraudulent URLs to ad objects, compelling users to click on links leading to suspicious or malicious websites. Through deceptive practices, such as cloaking or URL masking, fraudsters disguise the true destination of the ad click, redirecting users to unauthorized or potentially harmful webpages. We were able to successfully simulate URL misdirection by taking in files having fraudulent URLs. We are restricting our research into URLs which lead to phishing and defacement websites.

V. Mitigation techniques

A. *Chaperone and Disorientation Exploits*

In the previous section we discussed two immersive threats, the chaperone and disorientation threats, both the threats occurring due to the manipulation of the same chaperonr_info.vrchap file. Hence we believe that mitigating one threat successfully mitigates the other as well. We propose the use of digital signatures as a mitigation technique for these threats.

A digital signature is a cryptographic technique used to verify the authenticity and integrity of a digital message or document. It provides a way to detect tampering of files. Any modification to the JSON file will invalidate the signature of the file, indicating that the file has been tampered with. At the beginning we generate a digital signature for the initially defined chaperone_info.vrchap which would be the safe file before tampering because it contains the play area coordinates that the user has defined by providing a safe area tour in the initial start of the stream. For generating the digital signature of the JSON file we use SHA-256 hashing algorithm. We store a copy of this file in a NoSQL database like MongoDB. Whenever the VR environment is reloaded, the SHA-256 hash of the chaperonr_info.vrchap file is generated, and compared to the signature generated at the beginning. After the comparison, if the signatures are found to be different, we reload the file from the database, and reload the VR environment, and notify the users of the modification of the file and the change in the chaperone system.

B. *Buffer Overflow exploit*

The mitigation technique for Buffer overflow attacks involves 2 steps:

1) Mitigating the risks of the attack
2) Analysis to determine the attacker

To mitigate the risk of overflow encountered on uploading text files, we take advantage of the fact that, at any point, we have no need of displaying the entirety of text on the screen. So we have no need for storing the entire text string that we upload, since only a part of it can be realistically rendered on screen. Thus we can store the uploaded text file's contents in a string variable(which will be stored in heap memory since its size is determined at runtime), and read the string few characters at a time, and display them on the screen, by assigning the corresponding substring of the string variable to the textMeshPro text attribute dynamically during runtime. This effectively mitigates the attack. However, storing the entire text file's contents in RAM might also be a problem and a strain on the memory. This is where we use text compression algorithms like Burrows-Wheeler Transform, LZMA, or LZIP, and keep the compressed text file's contents in RAM, only expanding the portion of the text that needs to be displayed on the screen. This is a very quick set of operations that does not increase the latency and response time of the application.

To mitigate the risk of overflow encountered on uploading 3D models, we can use the different 3D model compression algorithms that reduce the number of polygons in the model that need to be rendered. For this we first send the obj file that is being uploaded by the user to the machine that is the host of the meeting. We assume that the host of the meeting uses a machine of very high RAM and processing capabilities. On receiving the obj model, the host uses the compression algorithms to compress the 3D model, and reduces its polygon count, and send the resultant OBJ file to every other user in the meeting, for synchronization among all the users. The 3D model compression algorithms are implemented using python's pymeshlab module.

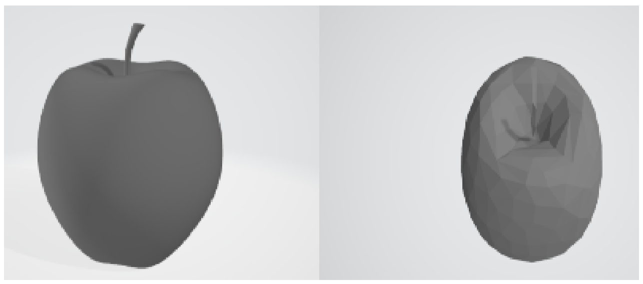


Fig 7: before and after compression of a 3d model to display in the VR environment.

However the above methods presented only deal with the first step of the mitigation technique. There is a need to perform behavioral analysis to determine the malicious user in the meeting. We believe this is necessary, as for this attack, the attacker must be one of the members of the meeting, and identifying such a person to prevent future malicious actions from him is necessary.

Most of the files which are malicious in nature are machine generated specifically to cause buffer overflow in systems. So, when the user uploads a file we check whether the file is a garbage file which is uploaded specifically for causing buffer overflow. To do this we use NLP, specifically we use TF-IDF and BOW approach to create a system which flags such files. We analyze the file contents with predefined contexts which the meeting host sets. By accurately detecting such threats, we enhance meeting security.

To develop an algorithm that performs behavioral analysis with the obj files for the 3d models as the input, we need to first understand how overflow exploits can be performed using 3D models. A malicious user who intends on performing an overflow exploit, can break up existing faces in the 3D model into several smaller faces, increasing the number of faces and therefore the number of vertices in the 3D model. This of course leads to a huge number of vertices having to be rendered in the application, increasing the size of the 3d model, without actually changing the appearance of the model, and the size of the obj file, thereby leading to a buffer overflow.

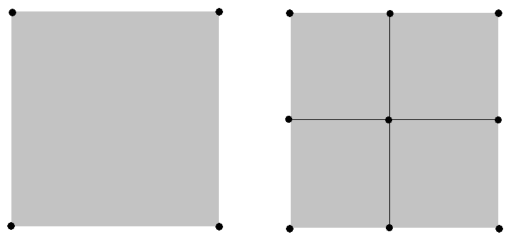


Fig 8: subdivided face of a cube leading to more vertices and faces to be rendered.

To flag models whose size has been increased by the method of subdividing each face into many subfaces, we can first make a simple observation of the fact that, on dividing faces into many sub faces, the number of faces in the model with the unit normal pointing in the same or nearly the same direction increases. Parsing through the OBJ file uploaded by the user, we can find the number of faces whose normals are the same, add them up and divide them by the total number of faces in the model, and check whether the division results in a value greater than a given threshold. But just doing the above step might result in a flag when shapes like sphere or the cylinder are uploaded. For this, we use the variance in the normals. Variance in statistics specifies the "spread" of the data points. In a sphere we can observe that the successive normals differ by a very small amount, which results in a low variance. If the size of the model is increased artificially by the method specified above, there end up being many faces in the model with the normals of these faces in the same direction, and instead of the gradual change in the direction of the normals of the faces that can be observed in a continuous 3d objects like a sphere, we can observe higher differences between the direction of the normals of the faces that have different normals in the model, thus increasing the variance of the normals. Using these parameters we can create a dynamic algorithm that analyzes the uploaded 3D model, and flags the user based on a certain threshold value for the variance.

Algorithm 1 Algorithm for analysis of users and uploaded 3d models for suspicious behavior

```
Input: .obj file of the 3d model uploaded by user$_i$
Output: flag variable for user$_i$
    Initialisation: flag user$_i$=0,count=0, threshold$_1$, threshold$_2$,file pointer f=open(.obj file), list l=[]
    LOOP Process
1: while $f\,!= NULL$ do
2:  l.push(unit normal of each face)
3: end while
    LOOP Process
4: for each $l_i$ in l do
5:     for each $l_j$ from $l_0$ to $l_i$ in l do
6:         if ($dotproduct(l_i, l_j) > 0.9$) then
7:             count=count+1
8:         end if
9:     end for
10: end for
11: if ($count/len(l) > threshold_1$ and $variance(l) > threshold_2$) then
12:     flag user$_i$=1
13: end if
14: return flag user$_i$
```

The above algorithm has a runtime complexity of $O(F^2)$ where F is the number of faces of the uploaded model.

Number of Faces	Time(s)
1000	2.217432022094726
2000	8.001606464385986
3000	17.59218621253967
4000	33.5003325939178
5000	57.25127840042114
6000	81.30241847038269
7000	98.89034914970398
8000	126.8605968952179
9000	163.9630579948425
10000	198.6718654632569

Table 1: Table shows the time taken in seconds for 3D model based behavioral analysis of users with respect to the number of faces in the 3d model.The time recorded is the total time taken for execution of the above algorithm 30 times,averaged over 100 runs. The timings are given for 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 2419 Mhz, 4 Core(s), 8 Logical Processor(s) and 8 GB RAM.

Hypothetically the algorithm is extremely parallelizable, as data parallelism can be performed on the list of normals of the faces.

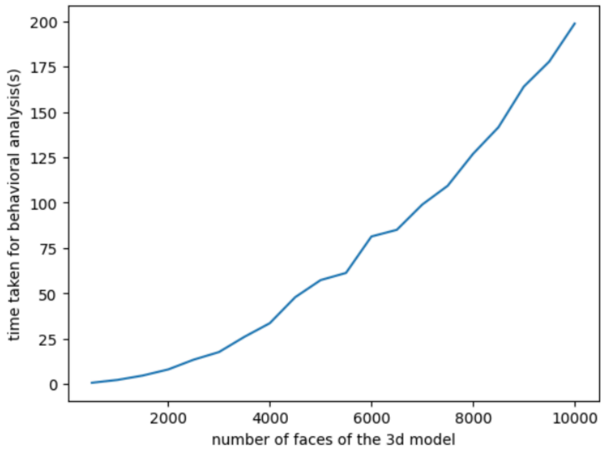


Fig 9: graph of time taken vs number of faces of 3d model

C. *Ad Fraud*

To address the prevalent ad fraud tactics like forced scaling through code and URL misdirection, we developed an AdGuard script for Unity and a static URL classifier. This script encloses an ad object within a larger box collider, preventing unauthorized resizing and distortion. This was made possible by using Unity's own feature in which a parent script can access and control the behavior of its children. Using this principle the script restricts the ad object from resizing beyond the limits of the box collider that encloses it. Whenever the advertiser uploads a script for the ad object, it goes through a check which ensures that the URL contained within the script is not, in any way, malicious. We used an ML model to achieve this. Trained on the dataset provided by the Canadian Institute for Cybersecurity, the Random Forest Classifier classifies the URLs into 5 categories. The categories being, Benign, Defacement, Phishing, Spam and Malware URLs. It is to be noted that this ML model gives a 95% accuracy in determining the type of URL. We employed a Random Forest Classifier model for this purpose, along with a set of eight parameters including url entropy, no of special characters, no of sub directories. The URL is classified by the model based on its characteristics without actually visiting the website, these characteristics include entropy, number of sub-directories and the number of numbers and special characters in the URL Through the combined functionality of the AdGuard script and ML-driven URL analysis, our approach offers robust protection against ad fraud in Unity. This proactive defense strategy ensures ad integrity and user safety, enhancing the reliability of digital advertising campaigns within the Unity development ecosystem.

VI. Results

During the course of our project, we successfully developed and applied security measures for the developed virtual reality conferencing application that efficiently prevent buffer overflow threats, immersive attacks, and ad fraud threats and offer comprehensive protection, with very little response delay. By controlling box colliders dynamically during ad spawns or uploads, we safeguard against unauthorized resizing and distortion of ads. We successfully developed techniques to prevent buffer overflow in the case of large file sizes. In addition, we have used NLP models as well as developed new algorithms to perform behavioral analysis on user uploaded data. We have integrated the above algorithms into the virtual conferencing application using a dedicated C# Library. Furthermore, the development of Dynamically Linked Libraries (DLLs) with implementations of the mitigation techniques for Chaperone and Disorientation attacks covers edge cases and provides versatile solutions for programmers.

VII. Conclusion

While our research into this topic suggests that considerable progress has been made in areas of Virtual reality and development of Virtual applications that provide extremely immersive experiences, it also suggests that much of research is yet to be done in the areas of security of Virtual reality applications. Our research and analysis highlights the need for a continued focus on several key areas:

- Developing robust threat detection and mitigation models: Real-time anomaly detection for suspicious VR user behavior and system activity is crucial for proactive defense.
- Prioritizing user privacy and data security: Implementation of robust encryption protocols, transparent data use policies, and secure user authentication methods.
- Promoting secure VR platform design and development**:** Integration of security considerations into the core design and development process of VR platforms and applications.

By addressing these crucial areas, we can ensure that the immersive world of VR becomes a safe and secure space for exploration, interaction, and innovation.

References

[1] Catalini C, Fons-Rosen C, Gaulé P (2020) How do travel costs shape collaboration? Manag Sci. https://doi.org/10.1287/mnsc. 2019.3381.

[2] J. A. de Guzman, K. Thilakarathna, and A. Seneviratne, "Security and privacy approaches in mixed reality: A literature survey," arXiv: 1802.05797, 2018.

[3] S. Li *et al.*, "An Exploratory Study of Bugs in Extended Reality Applications on the Web," *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, Coimbra, Portugal, 2020, pp. 172-183, doi: 10.1109/ISSRE5003.2020.00025

[4] P. D. Pazour, A. Janecek and H. Hlavacs, "Virtual Reality Conferencing," 2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), Taichung, Taiwan, 2018, pp. 84-91, doi: 10.1109/AIVR.2018.00019.

[5] P. Casey, I. Baggili and A. Yarramreddy, "Immersive Virtual Reality Attacks and the Human Joystick," in IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 2, pp. 550-562, 1 March-April 2021, doi: 10.1109/TDSC.2019.2907942.

[6] "GitHub Repository of aframevr/aframe," https://github.com/aframevr/aframe, 2020[45] "Issue tracker of aframevr/a frame on GitHub," https://github.com/aframevr/aframe/issues, 2020.

[7]"https://github.com/immersive-web/webxr/blob/master/explainer.md" The official Documentation for WebXR

[8] P. Casey, I. Baggili and A. Yarramreddy, "Immersive Virtual Reality Attacks and the Human Joystick," in IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 2, pp. 550-562, 1 March-April 2021, doi: 10.1109/TDSC.2019.2907942.

[9] Lee, H., Lee, J., Kim, D., Jana, S.S., Shin, I., & Son, S. (2021). AdCube: WebVR Ad Fraud and Practical Confinement of Third-Party Ads. *USENIX Security Symposium*.