Rise of Immersive Virtual Reality Malware and the Man-in-the-Room Attack

BLINDED, BLINDED, BLINDED

***Abstract*—In this work we present a primary account of the first Virtual Reality (VR) Worm & Botnet, and VR Man-in-the-Room (MitR) attack. We explore the applicability of old attacks in a new technological medium and the severity of the impact of these new attacks. We define abstract and formal foundations of VR Worms and MitR attacks against VR applications & platforms. We then devise our Proof of Concept (PoC) in the context of a widely used VR social application – Bigscreen. Unsurprisingly, our results illustrated a lack of security posture in the tested application, but more importantly, the novelty of the work is grounded in the severity impact of the malicious abuse of Immersive Virtual Reality, and the uniqueness of being virtually in the presence of others without their knowledge or consent. We share demonstrative attacking tools and used exploits. But we also focus on prevention, as we implement and publish a series of analytical tools, vulnerability signatures, and a dataset. Our work should inspire technical solutions to improve the state-of-the-art in VR security, socio-technical research in VR, and raise questions in the law and policy domains pertaining to VR security and privacy.**

***Index Terms*—Virtual Reality, Mixed Reality, VR Privacy, Security Analysis, Network Traffic Analysis, Penetration Testing, Reverse Engineering, Application Patching, Forensic Analysis, Responsible Disclosure, Bigscreen, Unity, Static Analysis Security Testing, Obfuscation, Deobfuscation, CodeQL.**

I. Introduction

VIRTUAL reality is an exciting and promising new technology, and has found a home in a variety of application including education, military and medical training and psychological treatment. Recent technological and manufacturing improvements in VR tilted the major use case towards entertainment by home consumers. Steam, both a partial subject of this study and popular game marketplace, estimates VR usage has doubled in 2018 [1].

While VR allows users to experience games, movies, and events with much greater presence and immersion than traditional mediums, the user's ability to interact with the Virtual Environment (VE) and peers elevates VR above other entertainment sources. This has benefited social interactions in particular, where not only are audio and video shared, but also a common space and simulated movements. In a 2018 survey, 77% of respondents reported an interest in more VR social interaction [2]. Many companies have brought social networking to VR, such as Facebook, Microsoft, vTime, VRChat and the target of this work Bigscreen [3]–[7].

Considering how lucrative social networks such as Facebook and Twitter can be, along with an anticipated uptick in social VR usage, an expected race would be fought to establish the dominant market foothold. Undoubtedly, this pressure may persuade developers to push their product to market without extensive security testing or a full understanding of the new technology. For this reason, we may expect small bugs or errors would be present in this new medium. We posit that the connectivity and medium of social VR applications drastically escalate the potential for exploitation.

Malware authors often target social networks due to the high degree of user connectivity. This allows malware to spread rapidly and have far-reaching effects [8]. Although created with no mal-intent, the Samy Worm (section II) exemplifies the potential for social network malware to propagate swiftly. From a single user, the worm spread to over 1 million victims in about 24 hours [9].

VR social networks are no exception to this possibility and offer a new and largely untested attack surface. Where traditional attacks might target intellectual property or aim to disrupt a user or infrastructure, VR has the potential to physically afflict the user. Furthermore, a wealth of information is often provided by the both the application or the VR system's own tracking, which can then be leveraged against the victim [10]. Simply put, a great deal can quickly go wrong for a large number of people if the security of VR is not explored.

Securing both social applications and VR systems is a challenging task. Any product that receives considerable exposure and handles complex interactions is subject to a wide range of potential attacks. Thankfully, web technologies are well seasoned (but not perfect) and attention has been turned towards securing VR, Augmented Reality (AR), and Mixed Reality (MR) alike. Although there have been efforts to secure both input and output of these systems (section II), countermeasures often do not make their way from theory to practice, as is often the case with emerging technology. Of note is that this work was partly inspired by the idea of how diseases spread in the real world upon person to person contact or being in the physical vicinity of others - we thought to ourselves, could current VR implementations support the spread of diseases in a synonymous fashion?

In our work, we deliver novel VR attack concepts supported by a model realisation. We further conduct an applied security analysis of a popular VR social application Bigscreen. Our work contributes the following:

- We are the first to coin and implement the *Man-in-the-Room* Attack
- We offer the primary account for an implemented Proof-of-Concept VR Worm & Botnet
- We conduct a deep security evaluation of a widely used immersive VR social application and show by example the impact of carried out attacks on VR users
- We impacted practice as both the Bigscreen has patched their application, and Unity Technologies documented our

newly discovered vulnerabilities and exploits – which could have affected millions of users
- We improve state of the art of vulnerability detection & prevention, as we implement and publish a series of analytical tools and vulnerability signatures as free and open-source software (FOSS)
- We create and publicly share a new and needed dataset for evaluation of Javascript (JS) Static Application Security Testing (SAST) focused on *jQuery*
- We share our demonstrative attacking tools and used exploits for research purposes as FOSS

The remainder of the paper is organised as follows. We present related and background information about social network worms and VR systems in section II. In section III we introduce novel attacks relevant to VR applications and platforms in general. Our research questions are specified in section IV. The adversarial model is defined in section V. We then describe our methodology and apparatus in section VI, followed by our security analysis. Findings are presented in section VII with a concrete realisation of MitR attack and VR worm. section VIII further documents how we improve current state of security and privacy. Possible mitigation and security suggestions are shared in section IX. Finally, we make concluding remarks, discuss results and point towards new directions in section X and section XI respectively.

II. Related Work & Background Information

A. Related Literature

A forensic study by [11] focused on VR social applications and documented forensically significant artifacts which can be used to reconstruct activities of VE participants. Several immersive virtual reality attacks including example ransomware which makes VR system unusable have been presented by [10]. Because users are immersed in VE, these attacks even managed to disorient and potentially physically hurt victims. Researchers were able to force movement of victims in real space without their knowledge or consent (Human Joystick Attack) [10].

Security and privacy of VR technologies was also investigated by [12]. The research exploited data from motion sensors inside VR equipment to infer a user's interaction with a virtual keyboard and touchpad. Researchers further investigated ways of interfering a user's interaction with input devices in VE based on stereo camera records of a user's body movement

Reconstruction of forensic artifacts from memory of VR devices was demonstrated by [13]. Testing an HTC Vive, researchers extracted and visualised data about VE, VR devices, room setup, and also about position and pose of a user.

Security challenges of MR application were highlighted by [14]. [15] identified and responsibly disclosed several security vulnerabilities in AR browsers. Some of these concerns are relevant to VR, too. [16] inspected security risks related to displaying information in AR in a scenario with malicious applications and [17] then designed an AR platform architecture with output security to limit abilities of individual applications based on a configured output policy.

With respect to the existing literature, our work is the first to exploit security vulnerabilities of a widely used VR social application, create a proof-of-concept VR botnet and VR worm, and implement the first Man-in-the-Room attack.

B. Background Information

We added this section to inform the reader on the topics of Cross-site Scripting (XSS), worms, Man-in-the-Middle (MitM) attacks, and Reverse Engineering (RE). Readers familiar with these concepts may want to skip this background information.

In this work, we exploit several vulnerabilities in the target application. Therefore, it is important that the reader is familiar with XSS. In general, an XSS vulnerability would be the result of poorly sanitised user input, which would in turn allow an attacker elicit script execution in the victim's browser [18]–[22]. This often results in the disclosure of sensitive data, such as passwords and cookies to the attacker.

A Persistent XSS attack has the potential to replicate its payload to every requester of the spoiled resource, thus allowing for the creation of a Worm [9]. This is exasperated by the usage of Asynchronous JavaScript and XML (AJAX) technology, whereby Hypertext Transfer Protocol (HTTP) requests may be made without user interaction or data transfer via WebSockets. Online Social Networks (OSNs) being a prime candidate for XSS Worm propagation, have given rise to several notable outbreaks such as the *Koobface*, *Mikey*, and *Samy* Worms [23].

The Samy Worm demonstrated characteristics similar to the Worm presented in this work. The Samy Worm, targeting MySpace pages was brought about by the protagonist purposefully infecting his own profile. The MySpace servers then delivered the payload to any requester of the infected page which then infected the victim's profile [24]. While newly infected profiles serve to propagate the Worm as well, [9] suggests that the centralized distribution of the payload prevents network congestion as caused by conventional Worms. This example Worm is limited to the exposure of infected users and still propagated at an alarming rate, however, our attack against the VR application is not limited by the social network topology.

While we can reasonably expect XSS vulnerabilities to continue to occur, the potential cost of a JavaScript Worm necessitates preventative measures. [18] proposed the first client side detection system, which monitors HTTP requests containing self-replicating payloads. [23] suggests deploying decoy profiles within an OSN, and [25] uses a proxy to tag and monitor HTTP content propagation.

Security analysis and penetration testing methodology is in detail examined by [26]. [27] further outlines security testing techniques and sets of tools useful for individual phases of a security assessment. Various approaches to security and forensic analysis are demonstrated by [28]–[30]. Work by [31] also includes implementation of PoC malware. [32], [33] published by Open Web Application Security Project (OWASP) offers a widely used web application penetration testing methodology and framework.

A MitM attack can be utilised for intercepting and spoofing secured network communication [34], [35]. Transparent prox-

ying allows for traffic interception without altering behaviour of analysed software [36]. Security analysis can utilise this for detecting possible information leak and for RE of application protocols. For example, [37] utilised findings from network traffic analysis of a popular mobile AR application; detected patterns were used to infer the locations of users.

Applications can often utilise forms of obfuscation and analysis protections. This applies to both compiled and interpreted programming languages [38]–[40]. [41], [42] demonstrate several approaches to code obfuscation. [43] provides a survey of tools for RE of C# & *.NET* and deobfuscation of JS which help to learn inner logic of an application.

C. Bigscreen Application

Bigscreen is a VR telepresence platform for social activities. It is intended not only for leisure (entertainment) activities like playing computer games, watching movies and hanging out, but also for productivity, work, meetings and collaboration (Figure 1). The application is available for Windows 7, 8.1, 10 operating systems via Oculus Home, Steam, and Microsoft Store.

Fig. 1. Productivity use cases of the Bigscreen. [7]

In Bigscreen's VE, every user is represented by an avatar, which copies moves of the user's head and hands in reality (Figure 2). Users access virtual rooms, they can create their own or join somebody else's. While in a room, a user can still use the computer via Head-mounted Display (HMD). Room participants can enable sharing their computer screens, computer audio, and microphone audio inside the room. The application also supports sending chat messages.

Fig. 2. VR avatars of room participants in the Bigscreen. [7]

Each room is characterised by the following properties: 1) Name, 2) Category (e.g. games, productivity, NSFW), 3) Description, 4) 3D Environment. Code of Conduct in the application stated the communication is Peer to Peer (P2P) and encrypted. 3D drawing and streaming ability of participants can be limited by a room admin using *locks*. Every room has a unique *Room ID* in the form of 8 alphanumeric characters (e.g. `room-9hckep83`). A room can exist as *public*, or *private / invite-only*. All public rooms are available on the application's main screen. Private rooms can be joined using a confidential *Room ID*, and no further authorisation is required.

III. Foundations of Man-in-the-Room Attack & VR Worm against VR Applications & Platforms

One of the approaches for building VR applications is the principle of virtual rooms. Users can usually create their own rooms or join existing ones. These virtual rooms serve as a place for interaction between users or with VE. Leisure activities, games, dating, productivity, collaboration, but even work meetings can take place in these rooms. VR also finds industrial applications. A private VR room may contain confidential information and interactions between users with a heightened privacy level than ordinary chat or video call. Users are immersed into this virtual world and lose awareness that they are still using a computer program. Their interactions are unrestrained. As the environment gets closer to what they know from the real world, they may assume that the real world's rules apply. Thus, users would not expect an invisible intruder (an invisible peeping tom) in their real living room, watching their activities and every move. Therefore, they may not expect this to happen in their VE either. Unfortunately, we proved that this assumption can be very wrong. We believe this intrusion can disturb people's privacy on a very personal level, compared to prior security work that has focused on non-immersive, or traditional computing environments.

We coined the concept of a novel MitR attack as follows. Assume that legitimate users communicate in a private virtual room. They can move in space, see actions of their avatars, and hear their voices. The attacker would then leverage security vulnerabilities in the employed VR software or infrastructure to gain unauthorised access to a private VR room. The attacker can then stealthily move around the virtual room while being invisible to everyone else in it. The concept of MitR means that the attacker is able to hear and see everything happening inside otherwise private VR rooms without victim's knowledge or consent. BLINDED has coined this novel privacy violation technique in VR a Man-in-the-Room Attack. *The possibility of this attach would critically affect security and privacy of VR.*

Our concept of a VR Worm is based on characteristics of regular computer worms, but with novel implications for VEs. A VR worm infects users of VR who become its hosts. A VR worm can then spread between users in a similar fashion when compared to a real disease – by contact with an infected person in virtual space. *We therefore illustrate that users should be careful who they meet virtually meet in VR.*

The MitR attack and VR worm described above are attack concepts which are applicable to VR applications and platforms in general. These attacks have clear goals – compromising a user's security and privacy. We further identify general prerequisites as main building blocks for a successful realisation of these attacks.

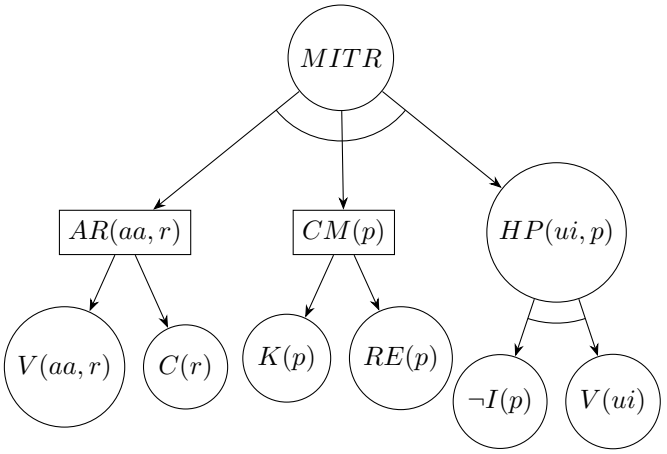

Fig. 3. Concept of a generic MitR attack against any VR application/platform defined as a proof tree, tree notation based on [44]. See Equation 1 for more details.

A successful outbreak of VR worms and their botnets throughout any VR platform depends on the following:

- **Vulnerable persistent environment** in a victim's VR application to host malicious code of a worm,
- **Functionality for duplication of a worm** to infect other vulnerable victims and spread through the platform,
- **Communication channel** for C&C protocol to allow control and monitoring of infected zombies in a botnet.

To perform a MitR attack in any selected VR application, the attacker is looking for the following:

- **Vulnerabilities in authentication & authorization** mechanisms or **credentials** to access protected private rooms,
- **RE of multimedia sharing protocol** to connect to victim's VR audio & video streams if the protocol in not well-known,
- **Lack of integrity checks** (or incorrect checks) to hide traces of attacker's presence in the VR room and also **vulnerabilities in User Interface (UI)** to remove traces in UI.

We further define MitR, it's sub-goals, and requirements formally in Equation 1 and Figure 3. Successful realisation of the attack against room r of targeted VR application/platform, which is using communication & multimedia sharing protocol p, authentication & authorization mechanism aa, and information is presented to user via UI component ui can be specified as a formula $MITR(p, ui, aa, r)$. Therefore, when we ask whether MitR attack is possible for a selected application t, we are looking for model $\mathfrak{M}_t$ and valuation v such that $\mathfrak{M}_t \models MITR[v]$ ($\mathfrak{M}_t$ satisfies $MITR$ with v). These formal definitions are later utilized in subsection VII-B.

$$
\begin{aligned}
MITR(p, ui, aa, r) = {} & R(r) \wedge P(p) \\
& \wedge AA(aa, r) \wedge UI(ui) \\
& \wedge AR(aa, r) \\
& \wedge CM(p) \\
& \wedge HP(ui, p) \\
MITR(p, ui, aa, r) = {} & R(r) \wedge P(p) \\
& \wedge AA(aa, r) \wedge UI(ui) \\
& \wedge (V(aa, r) \vee C(r)) \\
& \wedge (K(p) \vee RE(p)) \\
& \wedge (\neg I(p) \wedge V(ui))
\end{aligned}
\tag{1}
$$

where:

$R(x)$ = x is a VR room instance
$P(x)$ = x is communication & multimedia sharing protocol of VR application/platform
$AA(x, y)$ = x is authentication & authorization mechanism for a room y
$UI(x)$ = x is UI component of VR application/platform
$AR(x, y)$ = sub-goal to access room x with authentication & authorization mechanism y
$CM(x)$ = sub-goal to connect multimedia with protocol x
$HP(x, y)$ = sub-goal to hide presence in room from UI component x and from VE via y
$V(x)$ = x is vulnerable
$V(x, y)$ = x in combination with y is vulnerable
$C(x)$ = possession of credentials for x
$K(x)$ = details of x are publicly known
$RE(x)$ = successful Reverse Engineering (RE) of x
$I(x)$ = integrity checks for x are performed correctly

IV. Research Questions

Before the analysis was conducted, we had defined the following research question/objectives:

- Is a MitR attack possible/feasible in existing VR applications?
- Can malicious viruses/worms spread in VR like diseases in real life?
- Can old attack techniques have new consequences in VR?

For the evaluation of these research questions, we chose a widely used immersive VR social application called Bigscreen (subsection II-C).

V. Adversarial Model

We define the model such that the adversary seeks to both expand adversary controlled resources and harvest information from the target. Specifically, materials presented and conversations held while in private rooms.

The adversary does not require an environment or resources atypical of a normal user. The attack is crafted such that the adversary does not require prior knowledge of the target nor special network topology. For testing purposes, we include the assumption that the target is also VR capable and has or will launch the application.

VI. Methodology – Security Analysis of Bigscreen

We aim to deliver a concrete realisation of our coined MitR attack and VR Worm & Botnet which would be carried out against the Bigscreen application (subsection II-C). We also want to show by example the impact of carried out attacks on VR users. Our security analysis included penetration testing, which we categorise as: 1) External, 2) Black box, 3) Non-destructive, 4) Ethical, 5) In a controlled environment. The security analysis for this work was divided into several phases (Figure 4). These could be mapped to corresponding stages defined by [27] with some adjustments. Our phases can be briefly characterised as follows:

Reconnaissance focused on the examination of the Bigscreen and gathering of publicly available information.

Laboratory Setup & Tool Sets phase consisted of preparation of laboratory equipment and software tools based on identified areas of interest.

Security Analysis resulted in the outcome of our network traffic analysis, penetration testing, and RE of protocols and RE of the Bigscreen desktop application.

Exploit Development aimed at better assessment of the impact of discovered vulnerabilities. With identified security flaws, we crafted corresponding exploits.

Tool Construction covered the aggregation of discovered attacks and exploits into a comprehensive attacking tool.

Testing evaluated the success rate of exploits and of the tool according to defined scenarios which confirmed the high severity of our findings. Our findings were responsibly disclosed through appropriate channels.

A. Apparatus

Testing and analysis described in the study was carried out in a controlled laboratory environment similar to the work by [11]. Equipment used for VR included both the HTC Vive and Oculus Rift, the details are outlined in Table XV, Table XIII and Table XIV. Overview of tools useful for individual phases of the security analysis is presented in Table I.

B. Scenarios

We defined multiple scenarios for maintaining a systematic approach during the security analysis and during testing of the results. Scenarios refer to actions of hypothetical legitimate users (Alice, Bob) and attackers (Mallory, Trudy) as explained in Figure 5. Individual steps are defined in Appendix A. Scenarios cover standard usage of the application: 1) Passive stay in lobby 2) Creation of a public room 3) Creation of a private room 4) Conducting a private meeting 5) Transitioning between rooms

C. Reconnaissance – Phase I

We focused on Open Source Intelligence (OSINT) that could reveal inner parts of the Bigscreen system. We analysed job offers from the Bigscreen company and found out how does application's codebase look like.[1] Furthermore, Bigscreen's blog contained information about the application's updates and posts about their development.[2]

Previous forensic research [11] identified several artifacts left on the hard drive by the Bigscreen application. We uncovered that the application's UI elements were controlled by JS in a limited built-in web browser environment and the UI layer had bindings to the application's core layer.

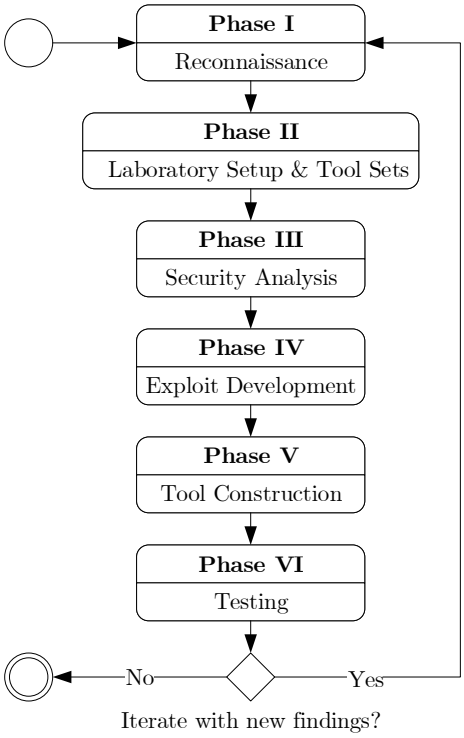

Fig. 4. State diagram of the phases of the process of carried out security analysis.

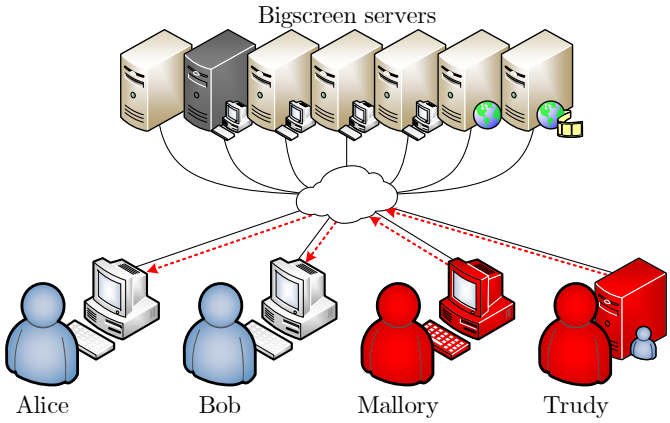

Fig. 5. Basic scenario for attacking the Bigscreen application. Alice and Bob are legitimate users of the application, each in a different location. Mallory is an attacker with maliciously patched Bigscreen application. Trudy is an attacker with developed C&C server capable of attacking Bigscreen users and controling created botnet. Mallory and Trudy aim at users of the application and do not attack Bigscreen servers.

[1] https://bigscreenvr.com/careers/
[2] https://blog.bigscreenvr.com/

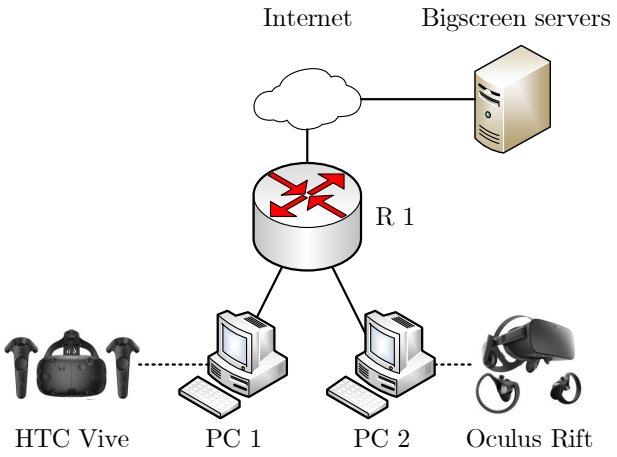

Fig. 6. Initial Experimental Setup

D. *Laboratory Setup & Tool Sets – Phase II*

The network topology for initial experiments is shown in Figure 6. This was later adjusted for specific experiments. The tools useful for the following security analysis are listed in Table I.

TABLE I
TOOLS FOR SECURITY ANALYSIS

Purpose	Software
network traffic analysis	mitmproxy, wifimitm, Scapy, Wireshark, Netfox Detective, NetworkMiner, *tools from [27, A-1]*
RE of C# and *.NET* environment	de4dot, Reflexil, ILSpy, dnSpy, JetBrains dotPeek, Progress® Telerik® JustDecompile
RE and deobfuscation of JS	JS Beautifier, JSNice, Prettier, Packer (unpacker) by Dean Edwards, de4js, ESDeobfuscate, JStillery, JSDetox, dCode Javascript Unobfuscator

E. *Security Analysis – Phase III*

The following part outlines a security and forensic analysis of Bigscreen's network traffic, a penetration testing of the application from the network side, RE of application's network protocols, and RE of Bigscreen desktop application. Results from this phase are summarised in Table XII and Table II.

a) MitM Attack against HTTPS & WSS: First network traffic analysis was carried out in the initial topology (Figure 6) according to defined scenarios (subsection VI-B). Even though the traffic capture contained some unencrypted Application Programming Interface (API) calls via HTTP, most of the traffic generated by the application was encrypted. This was circumvented using transparent proxy via *mitmproxy*, where a temporary Certificate Authority (CA) was configured and trusted by the VR workstations (Figure 7). This allowed us to decrypt Hypertext Transfer Protocol Secure (HTTPS) and Secure WebSockets (WSS) traffic.

b) Infrastructure Mapping: Throughout the network analysis phase, the application's network communications were monitored allowing us to create a map of Bigscreen's network infrastructure (Figure 7).

c) API Analysis: By analysing traffic between the application and Bigscreen's servers, we gained knowledge of individual endpoints. We observed the lack of authentication and encryption at several endpoints (Table XII). We uncovered that the Bigscreen application updated its UI by downloading it from the server using HTTP. Should the attacker get into the MitM position for the victim, they can spoof the UI files during download. Banlist including *uuid*, *reason* and *username* was also publicly available.

d) UI Layer Analysis: The downloaded UI files represented JS environment which communicates with Bigscreen's C# core layer locally via JS-C# function bindings. This UI layer also communicated with Bigscreen's servers using WSS. Obtained JS source code was obfuscated and minified, but after extensive analysis and with a significant effort we managed to reverse key parts. We later implemented a deobfuscator for this format (subsection VIII-C).

During analysis of JS-C# bindings, we discovered a critical security vulnerability in the C# Unity scripting API. A method `Application.OpenURL(url)` is dangerously capable of running programs, opening folders and files on the host computer. This method can also be used to automatically download and execute any payload (e.g. malware) on the host computer.

e) Signalling Protocol Reverse Engineering: By monitoring and studying the network data, we were able to reverse engineer Bigscreen's signalling protocol. This signalling channel was used to manage VR rooms and establish multimedia P2P channels. We observed WSS is encrypted, but used without authentication and mainly without authorisation. This meant that an encrypted, unauthenticated, and unauthorized message is able to manage any room selected by *Room ID*. This included changing settings and kicking users out of rooms.

f) Penetration Testing of the UI Layer from the Network Side: We were able to send a specially crafted message to the victim using the signalling channel. At the network message level, the attacker can set arbitrary values to username, room name, room description, and room category. The attacker can send a signalling message to the signalling server and the message is forwarded to room participants or users in application's lobby. The application did not perform proper sanitization of data received through encrypted signalling channel from the signalling server. *The Bigscreen application naively trusts the Bigscreen signalling server.* We have therefore discovered a XSS in room participant name, room description, room category, and room name.

g) Application Reverse Engineering: We managed to RE and decompile portions of the application and Dynamic Link Libraries (DLLs) into corresponding logic in C#. This allowed us to explore the inner structure of the application (Figure 8). We found out that DLLs are loaded without integrity checks. This made the unauthorised patching of the Bigscreen application possible (i.e. Application Crippling).

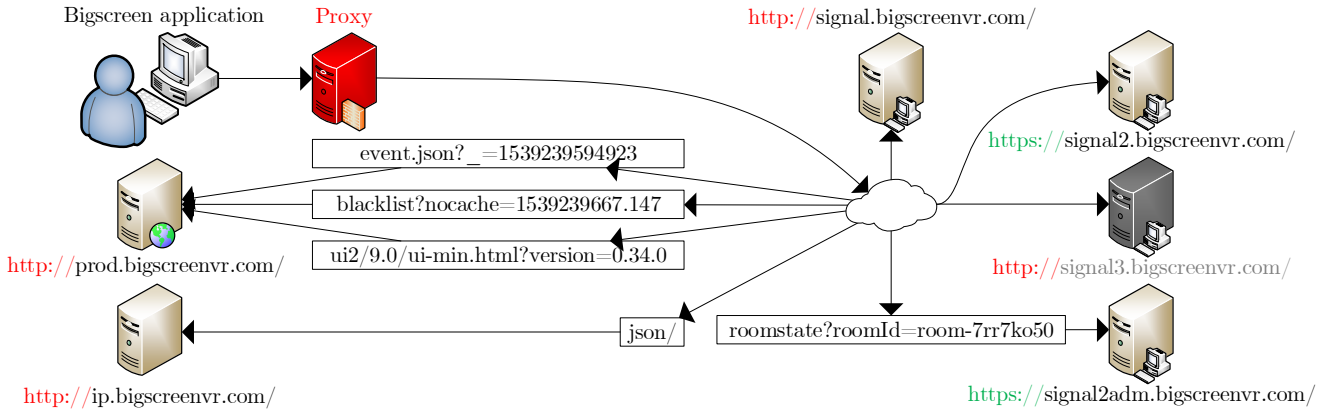

Fig. 7. Map of Bigscreen's Network Infrastructure.

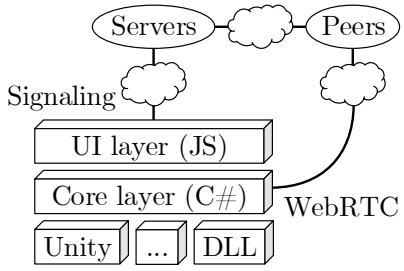

Fig. 8. Diagram of the Bigscreen application, which consists of several layers as explained in paragraph VI-E0g.

TABLE II
MAIN VULNERABILITIES OF THE APPLICATION

Vulnerability	Context	Severity
RCE via API call to `Application.OpenURL`	Unity engine	High
XSS in *user name*, *room name*, *room description*, and *room category*	UI layer	High
Information leak via RE of assemblies	Application core	Medium
Information leak via RE of obfuscated and minified JS source code	UI layer	Low
Patching DLLs without integrity check	DLLs	High
Lack of integrity, receiving data without sharing any VR state	WebRTC	High
Lack of authentication, connection from a custom application	WebRTC	Medium
Lack of authentication, connection from a custom application	Signaling channel	High

F. Exploit Development – Phase IV

In order to know the severity of identified issues, we focused on ways how malicious hackers could possibly abuse the Bigscreen application and put its users at risk.

a) Room State Polling: The `/roomstate` endpoint had no request limits. We developed a brute forcing script that could search for private *Room IDs*. However, the size of the target state space makes such a tool impractical. Dictionary attacks are not feasible, because *Room ID* consists of random 8 alphanumeric characters like `room-9hckep83`.

b) Automated Room Creation: Lack of authentication and authorisation in the signalling channel allowed us to continuously request the server to allocate resources for a new room. This could potentially lead to a Denial of Service (DoS) attack.

c) Kicking All Users from All Public Rooms: Forged signalling messages can kick any user out of any room. The only required information was the *Room ID* and identification of a given user. This is a possible DoS attack.

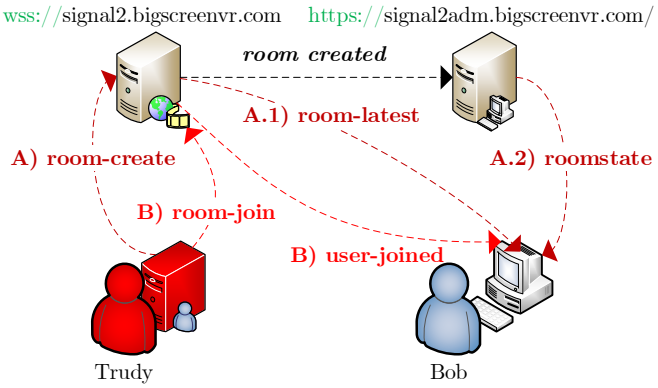

Fig. 9. Two possible paths of XSS attack over the network against Bigscreen users. On path A, the attacker creates a new public room with payload in room name, room description, or room category (`room-create`). Bob requests list of all public rooms which causes XSS in application of all users in the lobby (`room-latest`). Victim can also request details about selected room which also delivers the XSS payload (`roomstate`). On path B, the attacker sets payload as username and joins Bob's room (`room-join`). As soon as the attacker joins the room, XSS is executed in Bob's application (`user-joined`).

d) JS RCE Using XSS through Signalling Channel: XSS attack gains complete control over the UI layer of the victim's application. Two possible paths are illustrated in Figure 9. Attacking the lobby can affect all users of the Bigscreen application worldwide.

e) Eavesdropping Victim's Computer Screen, Computer Audio, and Microphone Audio: XSS attack can override victim's multimedia sharing. Our PoC WebRTC application was able to connect to legitimate Bigscreen application. With this exploit, the victim would unknowingly send their multimedia to the attacker.

f) Discovery of Private Rooms: The XSS attack can force the victim to leak ID of a created room. Room ID is sent to the attacker's C&C server.

g) Download & Execute Malware on Victim's Computer: Security flaw in Unity scripting API can be exploited to run programs, open folders and files on the victim's computer. The attack downloads and executes malware, as illustrated in Figure 10.

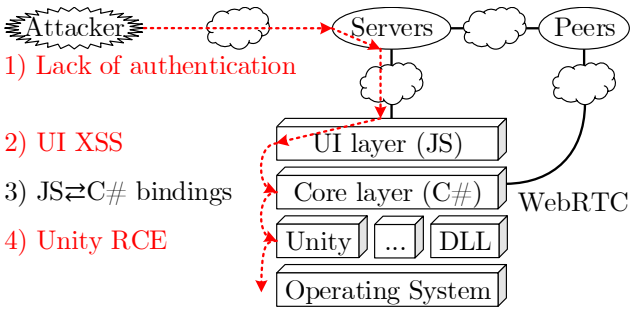

Fig. 10. Diagram of developed exploit to download and execute malware on victim's computer. Bigscreen application consists of several layers and communicates over network with its servers and peer users in the room. Attacker sends payload to servers which distribute it to users. Payload arrives to UI layer where it causes XSS attack. The attack propagates through bindings from UI to core layer. Application core then calls method from Unity with malicious payload which causes RCE. Attack escapes from the application, downloads malware and executes it.

h) Botnet and VR Worm Spreading through the Whole Bigscreen Community: Combination of several discovered vulnerabilities allowed for realisation of VR worm, implementation is described in section VII.

i) Man-in-the-Room Attack: A new cyber attack related to VEs has been successfully realised during this research (details in section VII).

G. *Tool Construction – Phase V*

To demonstrate our findings understandably and to the fullest, we decided to incorporate invented attacks into a single easy-to-use attacking tool. The tool is able to execute individual attacks and it also works as a C&C server which controls the entire botnet of infected Bigscreen applications. This tools acts as a dashboard with all relevant information and controls available. It offers functionality to release our first PoC VR worm. Malware delivery & execution can be done with a single button. Technologies incorporated in the implemented tool are sumarised in Table XVI.

Dashboard's functionality includes not only monitoring of all public rooms, but also discovery & monitoring of private rooms. The attacker can attack any room and take control over the Bigscreen application of any room participant. Dashboard can control multiple rooms at the same time using their *Room control* (Figure 11). Once the attacker joins a room, chat messages of room participants are eavesdropped and shown in *Room chat* panel (Figure 11). The attacker can covertly watch victim's computer screen, listen to computer audio & microphone audio using *Room participant* panel (Figure 11).

The tool monitors zombies and maintains connection with them. Each zombie can be controlled using corresponding *Zombie control* (Figure 11). Zombie's chat messages and application logs are continuously eavesdropped even after the zombie changes rooms.

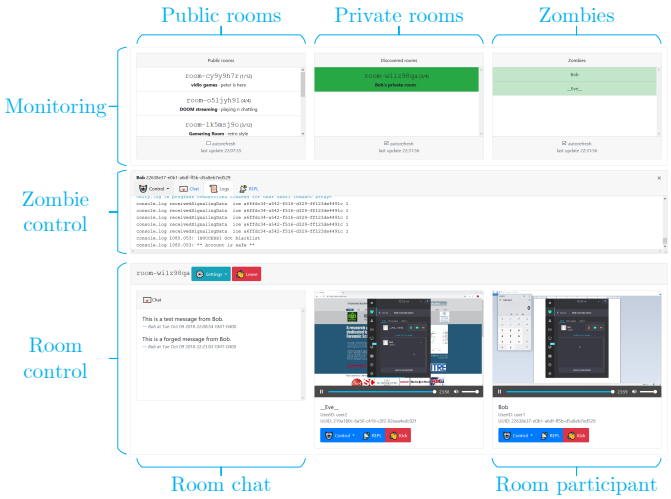

Fig. 11. Visualisation of the main parts of the C&C tool. *Zombie control* and *Room control* can be opened and closed for each controlled zombie and room. Each room can have multiple *Room participants*.

Clicking on the *Control* button opens the *Control menu* (Figure 12) which offers a variety of prepared attacks, most of them correspond to exploits summarised in Table VIII, Table XI and Table X. Another interesting attack is phishing (Figure 13), which is not related to RCE in Unity.

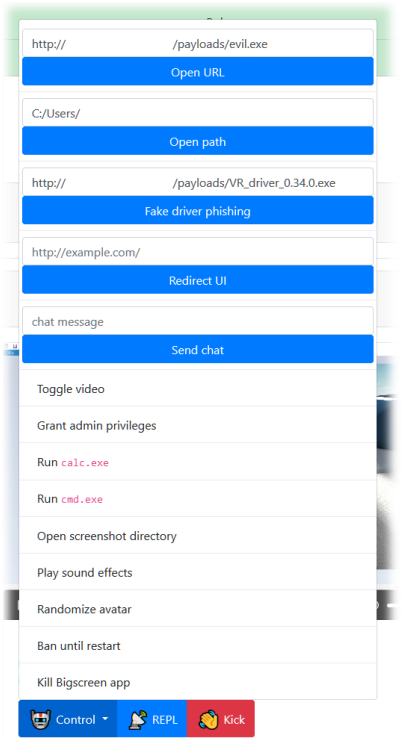

Fig. 12. *Control menu* gives the attacker ability to execute various attacks against selected victim/zombie. The menu is available from *Room participant* panel and similar menu can be opened from the *Zombie control* panel.

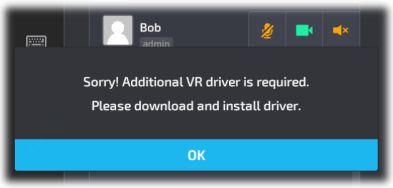

Fig. 13. *Control menu* also allows the attacker to execute phishing attack. Victim's Bigscreen application shows modal window asking the victim to install some driver (malware). Clicking OK button downloads the malware. This phishing is not related to RCE in Unity.

We have developed our basic C&C communication protocol to effectively control zombies from the C&C server (Figure 14). Mechanism for controlling infected Bigscreen applications is based on a combination of XSS vulnerability in UI layer and lack of authentication in signaling protocol (Figure 9).

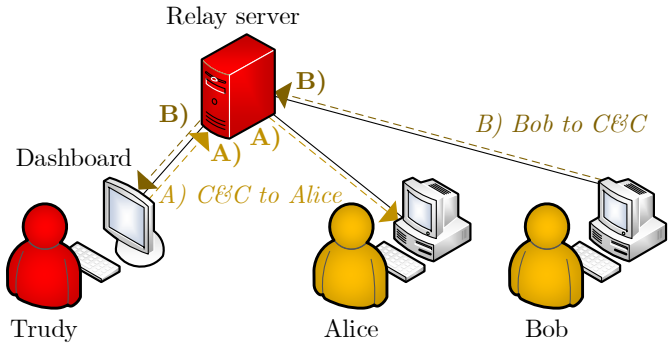

Fig. 14. Network diagram of developed relay server. Trudy uses C&C dashboard, which is connected to the relay server using WS as a client. Alice and Bob are both zombies already. Relay server forwards messages. Path A shows message sent from C&C to Alice, path B shows message sent from Bob to C&C.

To demonstrate possible malware outbreak, we developed a demonstrative malware, which does not cause any harm to an infected testing computer. It is used only to demonstrate that discovered attack could download and execute malware.

Implemented demonstrative attacking tool, including the dashboard, C&C server, relay server, and payloads to execute all exploits (subsection VI-F) is available online for research purposes.[3]

H. *Testing – Phase VI*

Testing was carried out according to defined scenarios (subsection VI-B and Appendix A). Note that experiments were carried out in a controlled laboratory environment. Attacks were limited to users and VR rooms created in our laboratory, as described in scenarios. Goals of the testing were mainly to:

- Validate success rate of individual attacks/exploits.
- Validate correct implementation of developed attacks in our C&C server.
- Validate feasibility of developed C&C protocol for controlling zombies.
- Describe critical impact of discovered vulnerabilities in *real-world* situations (scenarios).

Except for the separate phishing attack (Figure 13), all **tested attacks require no action from the victim**. Appendix B contains individual steps how the test cases were evaluated. Results are summarised in Table III.

TABLE III
TEST RESULTS BASED ON INITIAL SCENARIOS

Scenario	Test result
Passive stay in the lobby	Attack successful
Created public room	Attack successful
Created private room	Attack successful
Private meeting	Attack successful
Transition between rooms	Attack successful

VII. FINDINGS

Table II summarised main discovered vulnerabilities of the Bigscreen application. Infrastructure connections are in Table XII. Table VIII, Table XI, and Table IX contain individual developed exploits and attacks sorted according to the vulnerability they are based on. Advanced attacks require a combination of several discovered security flaws, as presented in Table X. Details are available in BLINDED.

We now explain how requirements of general attack concepts (section III) were fulfilled in the case of Bigscreen. We were able to use well-known techniques (e.g. XSS) for individual small steps. However, when chained together, they enabled these novel attacks with new and critical implications for VR users.

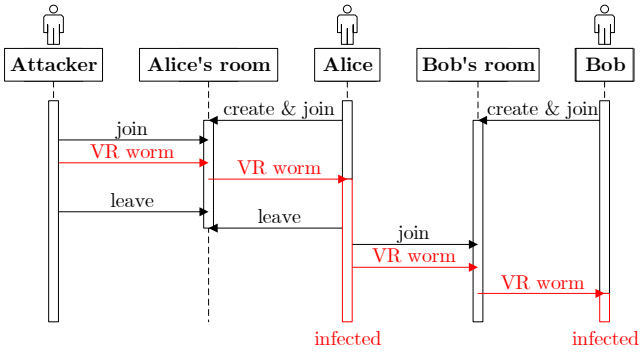

Fig. 15. Sequence diagram of of initial VR worm infection (in Alice's room) and propagation from one user to another when they meet in VR room (in Bob's room).

A. *VR Worm in Bigscreen*

a) Vulnerable persistent environment: Modifications by XSS attack can persist until application reset. Victims can, therefore, propagate the payload further.

b) Functionality for duplication of a worm: An attacker could modify the victim's name to also include an XSS payload, resulting in any future contact with other users to disseminate the payload and also modify their username, further circulating the attack (Figure 15). The principle of the worm is illustrated in Appendix D.

[3] BLINDED

c) Communication channel: When the victim gets infected by the worm, it becomes a zombie, reports to our C&C Server via WS established in context of UI layer (JS), and awaits commands (messages in Figure 14). With this exploit, it is possible to create a botnet of computers of the whole Bigscreen community and control them from the attacker's C&C Server.

B. Man-in-the-Room Attack in Bigscreen

a) Vulnerabilities in authentication & authorization: XSS through signalling channel was utilised to leak confidential Room IDs which were then used to access private rooms.

b) RE of multimedia sharing protocol: Due to the fact that the Bigscreen application loads DLLs without integrity checking (Table II), we managed to change the source code of selected libraries (patch) and the Bigscreen application still used these libraries. This allowed us to change selected behaviour. Our proof-of-concept patched Bigscreen application was able to connect with legitimate Bigscreen applications. This also gave us complete control over one end of audio/video/microphone/data streams (Figure 17).

c) Lack of integrity checks: We were able to hide our presence from UI of others using XSS payloads. Implemented Man-in-the-Room attack utilized multiple discovered attacks (Figure 17, Table X) in order to achieve invisibility in VR & UI. Victims would not have any information about the attacker being in their room. The attacker could see victims in VR, see screens of their computers, hear their audio/microphone (Figure 16).

We can now utilize formal definition from section III to specify how MitR attack was successful in Bigscreen application. Formula $MITR(p, ui, aa, r)$ represents a successful attack. For model $\mathfrak{M}_{Bigscreen}$, we consider predicate symbols from Equation 1 defined as follows, then we define valuations v_1 and v_2:

$$\begin{aligned}
R &= \{(\text{"}any\ public\ room\text{"}),\\
&\quad (\text{"}private\ room\ seen\ by\ infected\ user\text{"}),\\
&\quad (\text{"}unseen\ private\ room\text{"})\}\\
P &= \{(\text{"}proprietary\ over\ WebRTC\ \&\ WS\text{"})\}\\
AA &= \{(\text{"}public\ id\ knowledge\text{"},\\
&\quad \text{"}any\ public\ room\text{"}),\\
&\quad (\text{"}confidential\ id\ knowledge\text{"},\\
&\quad \text{"}private\ room\ seen\ by\ infected\ user\text{"}),\\
&\quad (\text{"}confidential\ id\ knowledge\text{"},\\
&\quad \text{"}unseen\ private\ room\text{"})\}\\
UI &= \{(\text{"}limited\ JS\text{"})\}\\
V &= \{(\text{"}limited\ JS\text{"})\}\\
V &= \{(\text{"}confidential\ id\ knowledge\text{"},\\
&\quad \text{"}private\ room\ seen\ by\ infected\ user\text{"})\}\\
C &= \{(\text{"}any\ public\ room\text{"})\}\\
K &= \{\}\\
RE &= \{(\text{"}proprietary\ over\ WebRTC\ \&\ WS\text{"})\}\\
I &= \{\}
\end{aligned} \tag{2}$$

$$\begin{aligned}
v_1 : p &\mapsto \text{"}proprietary\ over\ WebRTC\ \&\ WS\text{"}\\
ui &\mapsto \text{"}limited\ JS\text{"}\\
aa &\mapsto \text{"}public\ id\ knowledge\text{"}\\
r &\mapsto \text{"}any\ public\ room\text{"}
\end{aligned} \tag{3}$$

$$\begin{aligned}
v_2 : p &\mapsto \text{"}proprietary\ over\ WebRTC\ \&\ WS\text{"}\\
ui &\mapsto \text{"}limited\ JS\text{"}\\
aa &\mapsto \text{"}confidential\ id\ knowledge\text{"}\\
r &\mapsto \text{"}private\ room\ seen\ by\ infected\ user\text{"}
\end{aligned} \tag{4}$$

When we evaluate the formula with v_1 and with v_2, we see that $\mathfrak{M}_{Bigscreen} \models MITR[v_1]$ and $\mathfrak{M}_{Bigscreen} \models MITR[v_2]$.

VIII. Improving State of the Art of Vulnerability Detection & Prevention

During our research, we implemented a series of analytical tools and vulnerability signatures. We decided to publish them as FOSS to further contribute to the state of the art of vulnerability detection & prevention. MitR attack was possible due to vulnerabilities that could had been prevented. Developers and researchers can now use our tools to make other software more secure.

A. Dataset – Unsafe jQuery Method Calls

To evaluate abilities of our above-mentioned analyser and to compare it with other tools, we created a specific dataset. It consists of nearly 400 different samples of code and focuses on unsafe jQuery method calls. Samples are divided into two groups:

1) Context Awareness,
2) Taint & Data Flow Analysis.

The first group of samples includes code with various contexts, therefore trivial tools without context-aware parsers are eliminated from comparison. The second group includes code where ability to perform taint analysis and data flow analysis is required. This means the tool must understand whether and how are malicious data sanitised on paths from *sources* to vulnerable *sinks*; also how are these *sources* & *sinks* propagated through expressions.

B. jQuery XSS Static Analyser

XSS in Bigscreen's UI layer was caused by unsafe Hypertext Markup Language (HTML) manipulation with jQuery methods. HTML can be manipulated in a safe way by alternative approaches, so this was clearly a preventable development mistake.

To prevent such vulnerabilities, we implemented a static analyser for JS which can detect use of unsafe jQuery methods which are vulnerable to XSS attack. This analyser is available as a Command-line Interface (CLI) program, but also as a plugin for Coala static analysis system[4]. Plugins for Coala are

[4] https://coala.io/

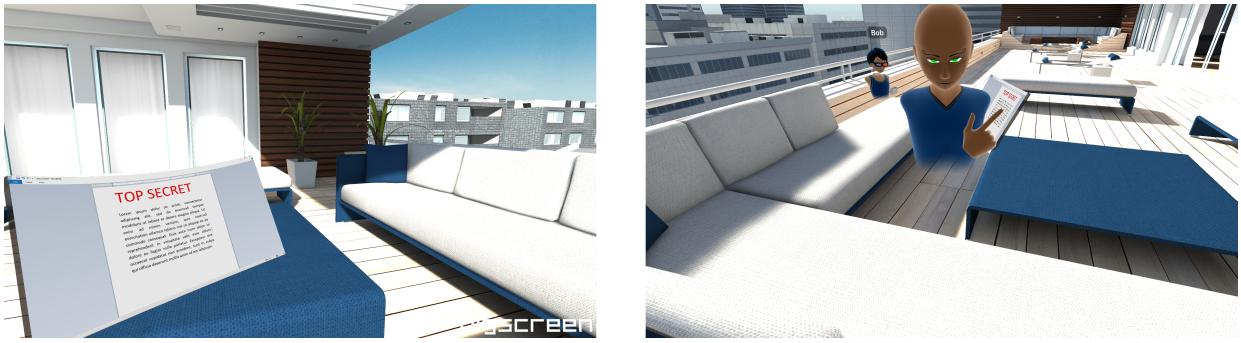

Fig. 16. Our novel Man-in-the-Room attack. Figure on the left shows the view of the user Bob, figure on the right shows attacker Mallory who is invisible while in the room with Bob. The attacker uses malicious version of the application, as shown in Figure 17.

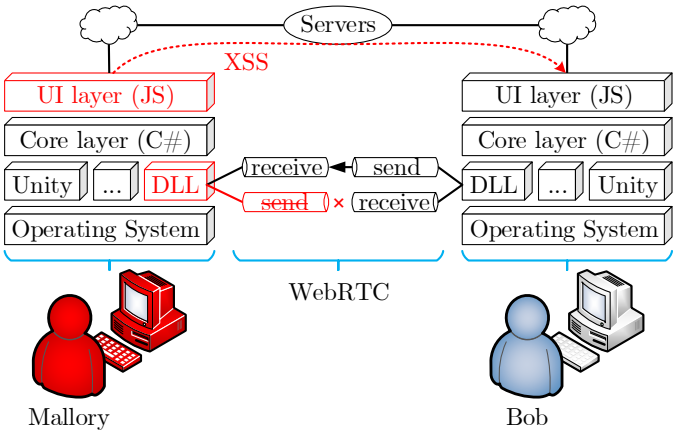

Fig. 17. The attacker Mallory uses patched (application crippling) version of the application which does not send VR state to other room participants and which also uses XSS payloads to hide traces of Mallory's presence in the room.

called bears[5] and this jQuery XSS Static Analyser is released as *JSjQueryXssUnsafeBear*. For example, it can be used as part of SAST stage of Continuous Integration (CI) (standalone or as a Coala plugin) by developers to make their software safer. The analyser is a FOSS and available from its repository[6].

To evaluate our tool, we use the above-described dataset. We also use this dataset to compare our tool with ESLint[7], ESLint with *jquery-unsafe* plugin[8]. Results of the evaluation of tools are shown in Table IV, Table V, and Table VI.

TABLE IV
CONFUSION MATRIX FOR EVALUATION OF ESLINT TOOL WITH THE DATASET.

		Context Awareness			Taint & Data Flow		
		P	N		P	N	
eslint	P	0	0		0	0	
	N	72	291	19,83%	12	12	50,00%
		0%	0%		0%	0%	

We can see that ESLint lacks the ability to detect jQuery unsafe calls.

[5]https://github.com/coala/coala-bears
[6]BLINDED
[7]https://eslint.org/
[8]https://github.com/cdd/eslint-plugin-jquery-unsafe

TABLE V
CONFUSION MATRIX FOR EVALUATION OF ESLINT TOOL WITH JQUERY-UNSAFE PLUGIN WITH THE DATASET.

		Context Awareness			Taint & Data Flow		
		P	N		P	N	
eslint +plugin	P	21	7	75,00%	0	7	0%
	N	51	284	15,22%	12	5	70,59%
		29,17%	2,41%		0%	58%	

TABLE VI
CONFUSION MATRIX FOR EVALUATION OF OUR JQUERY XSS STATIC ANALYSER WITH THE DATASET.

		Context Awareness			Taint & Data Flow		
		P	N		P	N	
jqxss	P	72	0	100%	0	12	0
	N	0	291	0,00%	12	0	100,00%
		100%	0%		0%	100%	

ESLint with a plugin cannot perform taint analysis or data flow analysis. It failed in all cases when data flow analysis was required to detect jQuerry access through several variables. In cases where taint analysis was required, because unsafe method was called with safe/sanitized value, it scored some true negatives. However, these true negatives are caused by the fact, that the tool does not know all unsafe methods. When an unsafe method unknown to it was tested, the code was marked as safe not because of understood sanitization, but because the tool thinks the tested method is safe. The tool has many false negatives as it cannot recognize more complex ways of calling unsafe methods, but its internal list of unsafe methods is incomplete. On the other hand, we can see some false positives, where it incorrectly detected method calls not related to jQuery.

Our analyser utilizes power of a parser and detailed knowledge of unsafe jQuery methods. This way we managed to eliminate false positives and false negatives in the context-aware part of the dataset. On the other hand, our tool does not aim to perform taint or data flow analysis. We decided that these techniques were out of scope during the implementation. Obviously, this sets some limitations for our tool. As we can see, it failed in taint & data flow section of the dataset because it does not implement required techniques. Results in this section are worse for our tool in comparison with ESLint with a plugin, but that's because ESLint with a plugin is not aware

of all unsafe methods, so it flags unknown ones as safe.

We can see that our tool reaches $100\%TPR$, $0\%FPR$, $100\%PPV$, $0\%FOR$, for part of the dataset focused on context awareness.

The dataset, test scripts, and results of the evaluation are publicly available[9].

C. JavaScript Array Ref Deobfuscator

Our deobfuscator is a CLI program which can revert *Array Ref* obfuscation format of JS. This format is characteristic by global array in the beginning of the file containing all values and method names used in the original source code. Obfuscated code then uses references to this global array instead of literals and methods. This makes manual analysis of the code very time-consuming, as production code can easily contain thousands of items in mentioned global obfuscation array. For example, UI layer of Bigscreen was obfuscated in this format. We have decided to share our deobfuscator with the professional community, it is published as FOSS and available from its repository[10].

D. BLINDED Fork

The above mentioned deobfuscator and static analyser need to be able to correctly parse JS source code and build corresponding Abstract Syntax Tree (AST). Use of a context-aware parser is essential for both minimizing false positives of static analysis, and delivering correct deobfuscation.

We decided to integrate the BLINDED library[11]. It primarily offers a JS minifier, but also serves as a library with a JS parser. Unfortunately, BLINDED parser was missing some features that our implemented tools needed. So we decided to create our fork[12] and extend it with needed functionality. Original library is available under MIT license, our fork is also available under MIT license and we plan to suggest merging updates from our fork to the main repository[13].

E. CodeQL Signature for Unity OpenURL Vulnerability

CodeQL allows for powerful SAST including taint analysis and data flow analysis. For example, GitHub is using it for batch scanning for vulnerabilities. However, CodeQL database creation is currently available only as a per-request service on LGTM[14]. In case of C# projects, CodeQL supports Microsoft visual Studio builds with msbuild. Unity-based projects can be compiled into resulting application with Unity's own build system. Unfortunately, CodeQL and LGTM lack direct support for projects based on Unity build system and buildless parser does not have `UnityEngine` API available. To leverage the power of CodeQL and achieve successful build by LGTM, we decided to use API placeholders for `UnityEngine` namespace, as the application compiled by LGTM is intended only for CodeQL analysis. This way, Unity-based projects can be built in LGTM to extract information for CodeQL queries.

We defined a CodeQL query and a build configuration which can detect vulnerability in Unity's `OpenURL` method. This is the vulnerability that we discovered and responsibly disclosed. The vulnerability signature is publicly available at the repository[15].

F. Unity OpenURL Exploit Demo

We implemented an example Unity application to fully demonstrate all the possible cases how the vulnerable `OpenURL` can be exploited. Our aim is to raise developers' awareness about this vulnerability. The demo exploit application is available from its repository[16].

IX. Mitigations & Suggestions

In this section, we present the mitigations we suggested to Bigscreen and Unity Technologies. The companies have used these measures to remedy the issues. However, these advices can be applied by any other company to improve security of their solution.

A. Bigscreen

Discovered weaknesses were caused by shortcomings & vulnerabilities in authentication, authorisation, encryption, data sanitization, integrity checking, or by a critical security vulnerability in 3rd party software (Unity engine). Individual flaws with smaller impact were chained together resulting in attacks with critical impact. Therefore, we suggest addressing the following.

a) Safe data manipulation and proper data sanitization: Because the application's UI is implemented with web technologies, it inherits security risks from the area of web applications. Several injection points for XSS existed due to unsafe HTML manipulation. We recommend using safe data manipulation and proper data sanitization at all times. We also recommend checking use of methods which can directly create and manipulate HTML without sanitizing data. One of the suggested solutions to this issue is to use some templating engine which would offer automatic escaping of data. Today's templating engines also often take care of *context-aware escaping*.

b) Secure authentication & authorisation: Both administrative activities and private rooms should have secure authentication & authorisation to determine the validity of requests. In order to join a private room, all that is required is the private `room-id`. We recommend introduction of user accounts and proper authentication & authorisation.

c) Cautious handling of insecure API: We suggest cautious handling of insecure API, especially proper sanitization of `url` parameter of the `Application.OpenURL` method from the Unity Scripting API.

d) Encrypted communication: Our team also strongly suggests using HTTPS communication instead of unencrypted HTTP.

[9] BLINDED
[10] BLINDED
[11] BLINDED
[12] BLINDED
[13] BLINDED
[14] https://lgtm.com/
[15] BLINDED
[16] BLINDED

e) Integrity checking: It is further suggested to ensure that the application and it's dependencies have not been modified. Methods like DLL integrity checking would be beneficial in this approach.

f) Enforcing VR state sharing: The application should monitor and enforce that all room participants correctly share information about their avatar and position in VE.

g) Brute force protection: The Bigscreen's server infrastructure should utilise brute force protection by for example enforcing limits on the number and frequency of requests made to the room status servers.

h) Confidential blacklist: We recommend changing the way of checking username against blacklist, because the whole blacklist should not be publicly available.

i) Removing development relics: Some of the debugging functionality and testing files have aided in our investigation. We recommend removing development relics and functionality unnecessary for production software.

B. Unity Scripting API

We are concerned about the ability of the `Application.OpenURL` method to run commands/programs and open directories/files on host systems (without scheme). We consider such functionality to be a severe security vulnerability. We suggest implementing parameter validations inside this API, which would prevent this issue.

We agree, it is reasonable for `Application.OpenURL` method to support various types of URL. However, some schemes might be unexpected for a developer. Therefore, we suggest considering their support. In case that support for schemes like for example `search-ms`, `ftp` and SMB is expected, we suggest one of following:

- Updating documentation with warning that developer has to conduct proper sanitization of parameter `string url` and also, warning about possible consequences would be very helpful.
- Updating `Application.OpenURL` method so that developers have to provide a second parameter in form of a scheme whitelist for a given method call.

X. Discussion & Conclusion

With respect to research questions, we proved with experiments that MitR attack is possible in an existing VR application and that VR Worm can spread between VR users like disease in real life. We showed that with new mediums, well-known attack techniques can evolve into new attacks with a novel impact. Compared to conventional applications, we posit that VR vulnerabilities are more privacy invasive. New VR systems collect a plethora of data such as a physical room structure, eye movements, hand and body movements etc. The technology presents new challenges that users, developers and companies are less experienced in, and for the users, it may be difficult for them to imagine that virtual worlds also present a new platform for spreading malware.

There aren't many platforms in which users may be educated about these new technologies. Most of the information people see come from the companies selling VR products, who naturally do not draw attention to potential risks. In addition, the products they bring to market are often published during development. We reacted to this by bringing our research to public attention in global media. We managed to publish the results together with explanation and recommendations for common users and the scientific community at large. Our hope is that it will help raise awareness of VR, its strengths and also its associated dangers.

The Bigscreen company accepted all recommendations we provided in the responsible disclosure and implemented appropriate security measures so that the described attacks may be mitigated.

Furthermore, Unity Technologies company addressed their issue by updating documentation of the dangerous method, as we suggested. After our responsible disclosure, several warnings have been included, so developers using the `OpenURL` method are now aware of its power and are instructed how to utilise it safely.

We help to prevent and detect vulnerabilities which were discovered and exploited during this research. We published several analytical or attacking tools, example exploits, evaluation dataset, and vulnerability signatures so that the professional community can collaborate on making software more secure.

Our work has impacted practice, and both a major VR application and a major development platform was improved significantly. The broad userbase of affected software is presented in Table VII.

We delivered a concept of novel VR attacks and we further identified key requirements for their realisation in any VR application or platform. Our work also presented an implemented primary account of the first Virtual Reality Worm, Botnet, and Man-in-the-Room attacks.

TABLE VII
Userbase

Software	Reach
Bigscreen Beta	over 500,000 users[17]
Unity	3,000,000,000 devices[18]

XI. Future Work

Our attacks were demonstrated on a single application. Future work should explore the automation of our methodical approach so that it may be expanded to other applications, as well as future VR systems. Published analytic tools and vulnerability signatures can be easily utilised. Future work should also focus on both legal and policy implications of our findings as VR technology gains more momentum.

References

[1] D. Heaney. (2019, January) Share of vr headsets on steam doubled in 2018. [Online]. Available: https://uploadvr.com/vr-steam-grew-2018/

[2] J. Koetsier, "Vr needs more social: 77% of virtual reality users want more social engagement," April 2018. [Online]. Available: https://www.forbes.com/sites/johnkoetsier/2018/04/30/virtual-reality-77-of-vr-users-want-more-social-engagement-67-use-weekly-28-use-daily/

[17] https://bigscreenvr.com/press/

[18] https://unity3d.com/public-relations

[3] Facebook, "Facebook spaces," 2019. [Online]. Available: https://www.facebook.com/spaces
[4] Microsoft, "Altspacevr." [Online]. Available: https://altvr.com/
[5] vTime Holdings Limited, "vtime." [Online]. Available: https://vtime.net/
[6] V. Inc., "Create and play in virtual worlds." [Online]. Available: https://www.vrchat.net/
[7] Bigscreen, Inc., "Press kit," Bigscreen, 10 2018, online. [Online]. Available: https://bigscreenvr.com/press/
[8] L. Danon, A. P. Ford, T. House, C. P. Jewell, M. J. Keeling, G. O. Roberts, J. V. Ross, and M. C. Vernon, "Networks and the epidemiology of infectious disease," *Interdisciplinary perspectives on infectious diseases*, vol. 2011, 2011.
[9] M. R. Faghani and H. Saidi, "Social networks' xss worms," in *2009 International Conference on Computational Science and Engineering*, vol. 4. IEEE, 2009, pp. 1137–1141.
[10] P. Casey, I. Baggili, and A. Yarramreddy, "Immersive virtual reality attacks and the human joystick," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2019.
[11] A. Yarramreddy, P. Gromkowski, and I. Baggili, "Forensic analysis of immersive virtual reality social applications: A primary account," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 186–196.
[12] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, and X. Fu, "I know what you enter on gear vr," in *Proceedings of IEEE Conference on Communications and Network Security (CNS)*, Washington, D.C., USA, 6 2019.
[13] P. Casey, R. Lindsay-Decusati, I. Baggili, and F. Breitinger, "Inception: Virtual space in memory space in real spacee – memory forensics of immersive virtual reality with the htc vive," *Digital Investigation*, 7 2019.
[14] F. Roesner, T. Kohno, and D. Molnar, "Security and privacy for augmented reality systems," *Commun. ACM*, vol. 57, no. 4, pp. 88–96, Apr. 2014.
[15] R. McPherson, S. Jana, and V. Shmatikov, "No escape from reality: Security and privacy of augmented reality browsers," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 743–753.
[16] K. Lebeck, T. Kohno, and F. Roesner, "How to safely augment reality: Challenges and directions," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '16. New York, NY, USA: ACM, 2016, pp. 45–50.
[17] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner, "Securing augmented reality output," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 320–337.
[18] F. Sun, L. Xu, and Z. Su, "Client-side detection of xss worms by monitoring payload propagation," in *European Symposium on Research in Computer Security*. Springer, 2009, pp. 539–554.
[19] S. Gupta and B. B. Gupta, "Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 1, pp. 512–530, 2017.
[20] Y. Wang, Z. Li, and T. Guo, "Program slicing stored xss bugs in web application," in *2011 fifth international conference on theoretical aspects of software engineering*. IEEE, 2011, pp. 191–194.
[21] A. Avancini and M. Ceccato, "Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities," in *2011 IEEE 11th international working conference on source code analysis and manipulation*. IEEE, 2011, pp. 85–94.
[22] A. Klein, "Dom based cross site scripting or xss of the third kind," *http://www. webappsec. org/projects/articles/071105. shtml*, 2005.
[23] W. Xu, F. Zhang, and S. Zhu, "Toward worm detection in online social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 11–20.
[24] W. H. Securtiy, "Cross site scripting worms and viruses, the impending threat and the best defense," 2006.
[25] V. B. Livshits and W. Cui, "Spectator: Detection and containment of javascript worms." in *USENIX Annual Technical Conference*, 2008, pp. 335–348.
[26] P. Herzog and M. Barceló, *The Open Source Security Testing Methodology Manual*, 3rd ed., Institute for Security and Open Methodologies (ISECOM), 12 2010, online. [Online]. Available: http://www.isecom.org/mirror/OSSTMM.3.pdf
[27] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, *SP 800-115: Technical Guide to Information Security Testing and Assessment*, National Institute of Standards and Technology, 9 2008.
[28] H. H. Alsaadi, M. Aldwairi, M. Al Taei, M. AlBuainain, and M. AlKubaisi, "Penetration and security of openssh remote secure shell service on raspberry pi 2," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2 2018, pp. 1–5.
[29] G. Dorai, S. Houshmand, and I. Baggili, "I know what you did last summer: Your smart home internet of things and your iphone forensically ratting you out," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. New York, NY, USA: ACM, 2018, pp. 49:1–49:10.
[30] X. Zhang, I. Baggili, and F. Breitinger, "Breaking into the vault: Privacy, security and forensic analysis of android vault applications," *Computers & Security*, vol. 70, pp. 516 – 531, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404817301529
[31] T. Haigh, F. Breitinger, and I. Baggili, "If i had a million cryptos: Cryptowallet application analysis and a trojan proof-of-concept," in *Digital Forensics and Cyber Crime*, F. Breitinger and I. Baggili, Eds. Cham: Springer International Publishing, 2019, pp. 45–65.
[32] M. Meucci, A. Muller *et al.*, *OWASP Testing Guide 4.0 - Release*. The OWASP Foundation, 9 2014, online. [Online]. Available: https://www.owasp.org/index.php/OWASP_Testing_Project
[33] A. van der Stock, B. Glas, N. Smithline, and T. Gigler, *OWASP Top 10 – 2017*, The OWASP Foundation, 2017, online. [Online]. Available: https://www.owasp.org/index.php/top10
[34] M. Vondráček, J. Pluskal, and O. Ryšavý, "Automation of MitM Attack on Wi-Fi Networks," in *Digital Forensics and Cyber Crime*, P. Matoušek and M. Schmiedecker, Eds. Cham: Springer International Publishing, 2018, pp. 207–220.
[35] ——, "Automated Man-in-the-Middle Attack Against Wi-Fi Networks," *Journal of Digital Forensics, Security and Law*, vol. 13, no. 1, pp. 59–80, 2018. [Online]. Available: https://commons.erau.edu/jdfsl/vol13/iss1/9
[36] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, "mitmproxy: A free and open source interactive HTTPS proxy," 2010–, [Version 4.0]. [Online]. Available: https://mitmproxy.org/
[37] G. Meyer-Lee, J. Shang, and J. Wu, "Location-leaking through network traffic in mobile augmented reality applications," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, 11 2018, pp. 1–8.
[38] S. D. Paola, "Advanced js deobfuscation via ast and partial evaluation (google talk wrapup)," Minded Security Blog, 10 2015, online. [Online]. Available: https://blog.mindedsecurity.com/2015/10/advanced-js-deobfuscation-via-ast-and.html
[39] G. Heyes, "Executing non-alphanumeric javascript without parenthesis," PortSwigger Web Security Blog, PortSwigger Ltd., 7 2016, online. [Online]. Available: https://portswigger.net/blog/executing-non-alphanumeric-javascript-without-parenthesis
[40] P. Palladino, "Brainfuck beware: Javascript is after you!" Blog Patricio Palladino, 8 2012, online. [Online]. Available: http://patriciopalladino.com/blog/2012/08/09/non-alphanumeric-javascript.html
[41] M. Mateas and N. Montfort, "A box, darkly: Obfuscation, weird languages, and code aesthetics," in *Proceedings of the 6th Digital Arts and Culture Conference, IT University of Copenhagen*, 2005, pp. 144–153.
[42] N. Montfort, "Obfuscated code," *Software Studies: A Lexicon*, pp. 193–199, 2008.
[43] M. Vondráček, "Security Analysis of Immersive Virtual Reality and Its Implications," Master's thesis, Brno University of Technology, Faculty of Information Technology, 2019.
[44] A. Kishimoto, M. Winands, M. Mller, and J.-T. Saito, "Game-tree search using proof numbers: The first twenty years," *ICGA journal*, vol. 35, pp. 131–156, 09 2012.

Appendix A
Scenarios Definition

Scenarios assume that Alice and Bob already have the Bigscreen application installed.

a) Passive stay in the lobby: Alice starts the application. She is on the first screen of the application, which is the lobby. List of all public rooms is downloaded from the servers and is displayed in the application's UI. Alice stays passively in the lobby for several seconds, then she terminates the application.

b) Created public room: Alice starts the application and enters the lobby. She creates & joins her new public room. She stays in the VR room for several seconds, then she leaves and terminates the application.

c) Created private room: Alice starts the application and she is in the lobby. She creates & joins private room. After several seconds, she leaves the room and terminates the application.

d) Private meeting: Alice starts the application. She waits in the lobby for few seconds and then creates & joins private room. Bob starts the application. Alice invites Bob, she shares her private room ID with Bob. Bob joins Alice's private room. Alice and Bob exchange few chat messages and interact in VR. Both participants leave the room after several seconds and both terminate the application.

e) Transition between rooms: Alice starts the application and creates & joins her public room. Bob creates & joins his public room. Alice stays for several seconds in her public room alone and then leaves. Alice joins Bob's public room. Alice and Bob spend several seconds together in the room and then they both leave and terminate the application.

Appendix B
Testing Based on Scenarios

Each test case starts by *C&C server setup procedure*, which consists of following steps. The attacker starts the relay server (Figure 14) and opens dashboard (Figure 11) which connects to the relay server using `dashboard-register` message. The dashboard connects to Bigscreen signaling servers and obtains list of public rooms for monitoring. The attacker ensures that testing malware is correctly prepared and available from the web file hosting server.

a) Passive stay in the lobby: The C&C server sends special signaling messages to Bigscreen signaling server which creates a public room with XSS payload hidden in the room name. This corresponds to path A in Figure 9. Alice downloads list of all public rooms. XSS payload is executed and Alice becomes a zombie in our botnet. Alice appears in zombie monitor in dashboard and Trudy opens *Zombie control*. Trudy forces Alice to download prepared testing malware and then forces Alice to execute it. Malware takes control of Alice's computer. **The attack was successful**, Alice was hacked and all she did was just opening Bigscreen application.

b) Created public room: Attacker Trudy has an overview of all public rooms in the dashboard. When Alice creates & joins her public room, the room appears in Trudy's dashboard. Trudy selects Alice's room and connects to it for eavesdropping using the dashboard. Alice thinks she is alone in the room. Trudy uses *control menu* to stealthily toggle Alice's video sharing. Trudy can see screen of Alice's computer now. She can take control of Alice's Bigscreen application and also download & execute malware on Alice's computer. Alice's has no suspicion that Trudy can see her screen. **Eavesdropping (attack) was successful.**

c) Created private room: The attacker Trudy starts attacking the lobby according to path A in Figure 9. As soon as Alice starts the application and lobby loads list of public rooms, she is attacked and her Bigscreen application becomes a zombie in our botnet. Alice creates & joins private room, but because she is zombie already, her application is automatically forced to leak confidential private room ID to Trudy's C&C server. The room ID is sent using `room-discovered` message of our C&C protocol. Alice's private room has just been discovered and it appears in monitor of private rooms in Trudy's dashboard. Trudy selects Alice's private room and connects to it for eavesdropping. Trudy toggles Alice's video sharing as well. Even though Alice created private room and she thinks she is alone in a secure room, Trudy can now see screen of Alice's computer. Trudy can take control of Alice's Bigscreen application and distribute malware, too. **The attack was successful.**

d) Private meeting: **This scenario tests also the novel MitR attack.** This test scenario includes another malicious actor called Mallory. Mallory uses our patched (Application Crippling) version of the Bigscreen application (Figure 17). Attackers Mallory and Trudy can communicate and coordinate the attack. However, this test scenario does not require Trudy and Mallory to be different people, one attacker could easily use the dashboard of C&C server and at the same time use the patched Bigscreen application. For clarity purposes, this test is described with both Trudy and Mallory. Trudy starts attacking the lobby. Alice starts the Bigscreen application, the lobby is opened, list of public rooms is loaded, Alice is attacked and becomes a zombie. Trudy can see Alice in a list of zombies. Trudy stops attacking the lobby. Alice creates & joins private room, room ID is leaked to Trudy. As described in the scenario, Alice gives Bob room ID and he joins Alice's private room. Trudy selects Alice's private room from list of discovered private rooms in the dashboard and connects to it for eavesdropping. Trudy can now control both Alice and Bob, she can also toggle their video sharing & see their screens. Trudy can distribute malware at this point. As Alice and Bob exchange chat messages, Trudy can see the messages in *Room chat* panel (bottom left part of Figure 11). Trudy can also spoof chat messages, for example impersonate Bob and write messages in his name. However, we want to see inside the VE of the VR room. Trudy shares obtained confidential private room ID with Mallory. Mallory joins Alice's room as invisible user. Alice and Bob have no idea that Mallory is with them in their private room. Mallory can move in virtual space, hear, and see everything what is happening in the room. This way, Mallory can literally look over their shoulders. **This attack including MitR attack was successful.**

e) Transition between rooms: This test is focused on the worm attacking lobby and spreading infection from one victim to another. Trudy starts attacking the lobby with VR worm. Worm infection was during testing limited to our testing users Alice and Bob. As Alice starts the application, she is infected with the replicating worm and becomes zombie. Trudy stops attacking the lobby. Alice creates & joins her new public room. Bob creates & joins his public room. Alice leaves her room and joins Bob's public room. **As soon as Alice meets Bob in virtual space, our VR worm duplicates and infects Bob.** This procedure is illustrated in Figure 15. Bob is now zombie, too. He also propagates the infection. Trudy can now see both Alice and Bob in list of zombies. Trudy can see that Alice's room no longer exists and that Alice and Bob are both in Bob's room. Trudy can eavesdrop on any room that Alice and Bob visit. From this point on, Trudy can take control

of every infected victim that Alice or Bob meet in VR while they carry the worm infection. Trudy can distribute malware to all these affected computers. **This attack including MitR attack was successful.**

APPENDIX C
OVERVIEW OF FINDINGS

TABLE VIII
DEVELOPED EXPLOITS BASED ON XSS VULNERABILITY

Category	Attack/Exploit	Severity
Botnet	Control infected applications from C&C server.	High
VR Worm	Spread a worm infection through the whole Bigscreen community.	High
JS RCE	Remotely execute any JS code in the UI layer of Bigscreen.	High
Privacy violation	Discover private rooms.	High
Privacy violation	Eavesdrop computer screen, computer audio, and microphone audio.	High
Privacy violation	Persistently eavesdrop victim's chat, even if they go to another room.	High
Phishing	Ask victim to install "required VR driver". This phishing is not related to RCE in Unity.	High
Privacy violation	Toggle video, audio, and microphone sharing.	High
Privacy violation	Change signaling servers of victim's Bigscreen application.	High
DoS	Remotely terminate victim's Bigscreen application.	Medium
Impersonation	Spoof chat messages.	Medium
Privilege escalation	Set selected user as room admin.	Medium
Phishing	Redirect Bigscreen's UI to any webpage.	Medium
Privacy violation	Gather all victim's logs.	Medium
DoS	Ban selected victim until restart.	Low
Miscellaneous	Change victim's avatar.	Low
Miscellaneous	Play sound effects from Bigscreen's UI.	Low

TABLE IX
DEVELOPED EXPLOITS BASED ON THE LACK OF REQUEST LIMITS

Category	Attack/Exploit	Severity
Privacy violation	Enumerate (brute force) room ID of private rooms.	Low

TABLE X
DEVELOPED EXPLOITS BASED ON THE COMBINATION OF XSS, RCE IN UNITY, AND APPLICATION PATCHING

Attack	UI XSS	Unity RCE	App. Patch	Severity
Man-in-the-Room.	✓	×	✓	High
VR Worm.	✓	×	×	High
Download & execute malware.	✓	✓	×	High
Run any program and open any folder or file.	✓	✓	×	High

TABLE XI
DEVELOPED EXPLOITS BASED ON THE LACK OF AUTHENTICATION & AUTHORIZATION IN THE SIGNALING CHANNEL

Category	Attack/Exploit	Severity
DoS	Kick any user from any room.	High
Privilege escalation	Change room's settings (VR locks).	Medium
Resource exhaustion	Automatically create new rooms.	Low

TABLE XII
NETWORK CONNECTIONS

Endpoint	Enc.[19]	A. & A.[20]
http://prod.bigscreenvr.com	×	×
http://signal.bigscreenvr.com	×	×
http://ip.bigscreenvr.com	×	×
https://signal3.bigscreenvr.com	*Out of order*	
https://signal2adm.bigscreenvr.com	✓	×
https://signal2.bigscreenvr.com	✓	×
wss://signal2.bigscreenvr.com	✓	×
P2P WebRTC channels	✓	×

APPENDIX D
JAVASCRIPT WORM EXAMPLE

Minimal example of self-replicating XSS payload in variable `NAME` which can be used for the VR worm.

```
function worm(){
  /* payload here */;
  NAME='<sc'+'ript>'+worm.toString()+
  ';worm();</sc'+'ript>John';
};
worm();
```

APPENDIX E
HARDWARE AND SOFTWARE DETAILS

TABLE XIII
APPLICATIONS

Application	Version
Bigscreen Beta	0.34.0
Oculus App	1.36.0.215623
Steam	1549129917
SteamVR	1.2.10

TABLE XIV
VIRTUAL REALITY DEVICES

Device	Component	Firmware
Vive	Headset Vive MV HTC	1462663157
	Base HTC V2-XD/XE	436
	Base HTC V2-XD/XE	436
	Controller MV HTC (x2)	1533720215
Rift	Headset Rift	709/b1ae4f61ae
	Sensor (x2)	178/e9c7e04064ed1bd7a089
	Left Touch	f3c65f7a5f
	Right Touch	f3c65f7a5f

[19]Encryption
[20]Authentication & Authorization

TABLE XV
SYSTEM DETAILS

Device	Details
Processor	Intel Core i7-6700 CPU
System Type:	64-bit OS, x64 processor
Graphics Card	NVDIA GeForce GTx 1070
Manufacturer	iBUYPOWER
Installed Memory (RAM)	8.00 GB
Operating System	Windows 10 (10.0.0.17134)

TABLE XVI
TECHNOLOGIES OF THE C&C SERVER

Technology	Reason
GUI	Easy-to-use dashboard
Video & audio	Eavesdropping on victim's microphone audio, computer audio, and computer screen
HTTP & HTTPS	Interaction with Bigscreen servers
WSS	Communication using Bigscreen's signaling protocol
WebRTC	Reception of P2P multimedia streaming
C&C protocol	Control and monitoring of zombies in botnet
File hosting	Malware distribution to victims