Rise of the Metaverse's Immersive Virtual Reality Malware and the Man-in-the-Room Attack & Defenses

Martin Vondráček[a], Ibrahim Baggili[b,*], Peter Casey[c], Mehdi Mekni[d]

[a] *Faculty of Information Technology, Brno University of Technology, Božetěchova 2/1, 612 00 Brno, Czech Republic.*
[b] *Louisiana State University*
College of Engineering & Center for Computation and Technology, Baton Rouge, LA 70803 USA
[c] *University of New Haven, West Haven, CT 06516 USA*
[d] *University of New Haven, Laboratory for Applied Software Engineering Research (LASER), West Haven, CT 06516 USA*

Abstract

The allure of the metaverse along with Virtual Reality (VR) technologies and speed at which they are deployed may shift focus away from security and privacy fundamentals. In this work we employ classic exploitation techniques against cutting edge devices to obtain equally novel results. The unique features of the Virtual Reality landscape set the stage for our primary account of a new attack, the Man-in-the-Room (MitR). This attack, realized from a vulnerable social networking application led to both worming and botnet capabilities being adapted for VR with potential critical impacts affecting millions of users. Our work improves the state-of-the-art in Virtual Reality (VR) security and socio-technical research in VR. It shares several analytical and attacking tools, example exploits, evaluation dataset, and vulnerability signatures with the scientific and professional communities to ensure secure VR software development. The presented results demonstrate the detection and prevention of VR vulnerabilities, and raise questions in the law and policy domains pertaining to VR security and privacy.

Keywords: Emerging technologies; Network-level security and protection; Network communications; Network Protocols; Protection mechanisms; Quality analysis and evaluation; System issues; Security and Privacy Protection; Authentication; Communications Applications; Artificial, augmented, and virtual realities; Virtual reality; Security and Protection; Invasive software (viruses, worms, Trojan horses); Unauthorized access (hacking, phreaking);

Virtual Reality (VR) aims to create "immersive, interactive, and imaginative" simulations for the user through visual, haptic, and auditory output [1, 2]. It is produced by a computer system that emulates realistic scenes to provide convenience for users to experience the virtual world. As an emerging human-computer interaction technology, VR uses several systems including voice input/output, motion sensing, network communication, computer graphics, and wide-angle stereo display systems. The applications of VR are gradually expanding, including but not limited to entertainment, education, healthcare, and military [3]. VR headsets have existed since the 1960s, one of the first devices that could be considered VR Head-mounted Display (HMD) was presented by [4]. Commercial applications of VR can be dated back to 1980s [5], where we can also find steps towards VR social applications such as VPL's *Reality Built for Two* [6, 7]. Affordable VR equipment is fairly recent to the consumer market [8], a headset with fully-embodied VR capabilities—the Oculus Rift Development Kit 1—became available in 2013 and its consumer version was released in 2016 [9]. The VR market has been growing ever since, with revenues projected to grow from $12B to a $100B in the next five years [10].

Recent technological and manufacturing improvements in VR tilted the major use case towards entertainment by home consumers. The Steam platform[11], both a partial subject of this study and popular game/application marketplace, estimated VR usage has doubled in 2018 and number of monthly-connected VR headsets reached 2 million in 2020 [12, 13, 14]. Furthermore, demand for VR games and applications grew with sales up 71% in 2020 [15]. And the excitement for VR continues as Steam reported that VR userbase grew 11% with unique play sessions up 22% in 2021 [16].

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 [17].

*These authors contributed equally to this work
Email addresses: vondracek.mar@gmail.com (Martin Vondráček), ibaggili@lsu.edu (Ibrahim Baggili), pgrom1@unh.newhaven.edu (Peter Casey), pgrom1@unh.newhaven.edu (Mehdi Mekni)

Many companies have brought social networking to VR, such as Facebook Horizon (previously known as Facebook Spaces), AltspaceVR by Microsoft, vTime, VRChat, and also Bigscreen, which was used for Proof of Concept (PoC) in our research [18, 19, 20, 21, 22]. At the time of writing, companies have also started exploring building the *metaverse*, a virtual universe.

Considering how lucrative social networks 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 may pressure developers to push products to market without extensive security testing or a full understanding of the new technology. For this reason, we may expect 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 malware exploitation, especially with the establishment of the *metaverse*. Malware authors often target social networks due to the high degree of user connectivity. This facilitates its rapid spreading and has far-reaching effects similar to infectious diseases in real life [23]. Although created with no mal-intent, the Samy Worm, described in Section 1, exemplifies the potential for social network malware to propagate swiftly. From a single user, the worm spreads to over 1 million victims in about 24 hours [24].

VR social environments are no exception to this possibility and offer a new and largely untested attack surface. While 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 both the application and the VR system's own tracking, which can then be leveraged against the victim.

In our work, we deliver novel VR attack concepts supported by a model realization. Our attacks are generalizable since we target core technologies widely used in VR systems. This is part of our VR security research and we build on top of our previous results [25], [26], [27]. Parts of the paper cover results of our research done in 2018 [28]. Our current work contributes the following:

- We are the first to implement the *VR Man-in-the-Room* Attack. Two of the authors coined and predicted this attack in previous work [26].
- We offer the primary account for an implemented *VR-specific Worm & Botnet*.
- We improve state of the art of *automated vulnerability detection & prevention*, as we implement and publish a series of analytical tools and vulnerability signatures as free and open-source software (FOSS).
- We conduct a deep *security analysis* of a widely used immersive VR social application and show by example the severe impact of carried-out attacks on VR users.
- We *impacted practice with responsible disclosure* as both the Bigscreen has patched their application, and Unity Technologies documented our major discovered vulnerabilities and exploits .
- We share our demonstrative attacking *tools and exploits* for research purposes as FOSS.
- We propose a new *verification and validation dataset* for Javascript (JS) Static Application Security Testing (SAST) focused on jQuery.

The remainder of the paper is organized as follows. Section 1 provides an overview on related work and background information about VR systems and social network worms. In Section 2, we present our novel attacks relevant to VR applications. Section 4 describes our methodology, apparatus, and scenarios followed by our security analysis. Section 4.8 outlines our findings supported by a concrete realization of Man-in-the-Room (MitR) attack and VR worm. Section 4.9 summarizes potential mitigation and security suggestions. In addition, Section 5 documents the way we improved the state-of-the-art of vulnerability detection & prevention. Finally, Section 6 and Section 7 provide concluding remarks, discuss results, and explore new research directions.

1. Related Work

1.1. Security & Privacy in VR

The security and privacy of VR technologies was investigated by [29]. 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. Moreover, several immersive VR attacks including ransomware which makes VR system unusable have been previously created by authors of this work [26]. Because users are immersed in the VE, these attacks managed to disorient and potentially physically hurt victims. It was possible to force movement of victims in real space without their knowledge or consent (Human Joystick Attack) [26]. Authors of this work also presented forensically significant artifacts which can be used to reconstruct activities in a VE [25]. We have further shown the ability to visually reconstruct a VE from memory artifacts such as devices, room setup, location, and a user's pose [27].

Security challenges of Mixed Reality (MR) applications were highlighted by [30]. [31] identified and responsibly disclosed several security vulnerabilities in a selection of Augmented Reality (AR) frameworks. Some of these concerns are shared with VR technologies. [32] inspected security risks related to displaying information in AR in a scenario with malicious applications. In addition, [33] designed an AR platform architecture to limit abilities of individual applications based on a configured output policy. A review on security threats in MR was presented by [34], while a survey of proposed protection mechanisms for MR was done by [35]. Study by [36] proposed a generic framework for MR applications. Work by [37] focused on VR security and privacy perceptions by conducting interviews with VR users and developers, and by making a survey of VR privacy policies.

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. It is also the first work to explore defenses against such attacks.

1.2. Security Background

Security analysis and penetration testing methodologies are detailed in The Open Source Security Testing Methodology Manual (OSSTMM) 3 [38]. *SP 800-115* from National Institute of Standards and Technology (NIST) further outlines security testing techniques and sets of tools useful for individual phases of a security assessment [39]. Various approaches to security and forensic analysis are studied by [40, 41, 42]. Work by [43] also includes implementation of PoC malware. A widely used web application penetration testing methodology and framework has been published by Open Web Application Security Project (OWASP) [44, 45].

In regard to network communication analysis, a Man-in-the-Middle (MitM) attack is used to intercept and spoof network traffic [46, 47, 48]. Security analysis can utilize this technique to detect possible information leak and for Reverse Engineering (RE) purposes. For example, authors in [49] utilized network traffic analysis techniques of a popular mobile AR application. They applied RE to extracted data, detected patterns, and inferred users' locations.

RE techniques exploit the weak forms of obfuscation and analysis protections that software applications typically utilize. This applies to both compiled and interpreted programming languages [50, 51, 52]. Authors in [53, 54] present several approaches for code obfuscation. Our work in [55] provides a survey of tools for RE of C# & *.NET* and deobfuscation of JS which help to learn inner logic of an application.

In addition to the use of the MitM attack, we also propose a worm whose characteristics are similar to Samy Worm (SW). SW targeted MySpace pages where the protagonist purposefully infected his own profile. The MySpace servers then delivered the payload to any requester of the infected page which then infected the victim's profile [56]. While infected profiles serve to further propagate the Worm, authors in [24] suggested that the centralized distribution of the payload prevents network congestion as caused by conventional Worms. Although this Worm was limited to the exposure of infected users, it propagated at an alarming rate. Unlike SW, our attack against the VR application is not limited by a given Online Social Network (OSN) topology.

The management of worm infections requires preventative measures. Authors in [57] proposed a client side detection system which monitors Hypertext Transfer Protocol (HTTP) requests containing self-replicating payloads. Other preventive measures suggest the deployment of decoy profiles within an OSN [58] or the use of a proxy to monitor HTTP content propagation [59].

1.3. Bigscreen Application

Bigscreen is a VR platform for social activities. It is intended not only for entertainment activities like playing computer games, watching movies and just hanging out, but also for professional productivity and remote collaboration purposes (see Figure 1). Bigscreen is available for Windows operating systems via Oculus Home, Steam, and Microsoft Store.

Figure 1: Productivity use cases of the Bigscreen [22].

In Bigscreen's VE, every user is represented by an avatar, which copies moves of the user's head and hands in reality as illustrated in Figure 2. Users create and access virtual rooms, control resources via HMD, and share their computer screens, computer audio, and microphone audio. Bigscreen supports messaging services among the participants.

Figure 2: VR avatars interacting in a virtual room in Bigscreen. [22]

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 authorization is required. Application's Code of Conduct stated the communication is Peer to Peer (P2P) and traffic is encrypted.

2. Man-in-the-Room Attack & VR Worm

2.1. Concepts Definition

A paramount approach for building VR applications is the principle of virtual rooms. They serve as a space for interaction between users and with the VE. 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 are expected to 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, VR users assume that the physical world's social and privacy rules apply. Thus, users would not expect an invisible intruder (an invisible virtual *Peeping Tom*) in their real living room, watching their activities and every move. 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 defined the novel concept of 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 leverage security vulnerabilities in the VR platform (or in the VE) to gain unauthorized access to a private VR room. The attacker would then manage to 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 authorization.

The concept of a VR Worm we propose is based on common characteristics of computer worms, but with novel implications for VEs. A VR worm infects users of VR who become its hosts. It can then spread among users sharing the same virtual room. This could lead to pandemic situation inside virtual spaces. We illustrate why users should be careful who they meet in VR. We outlined our threat/adversary model in Section 3.

2.2. Formal Modeling

The conducted study is motivated by the following research questions:

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

The MitR attack along with the VR worm we propose are novel concepts specific to security and privacy in VR applications. In our work, we further identify general prerequisites as main building blocks for a successful realization of these attacks. A successful outbreak of VR worms and their botnets throughout any VR platform must meet the following requirements:

- **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,

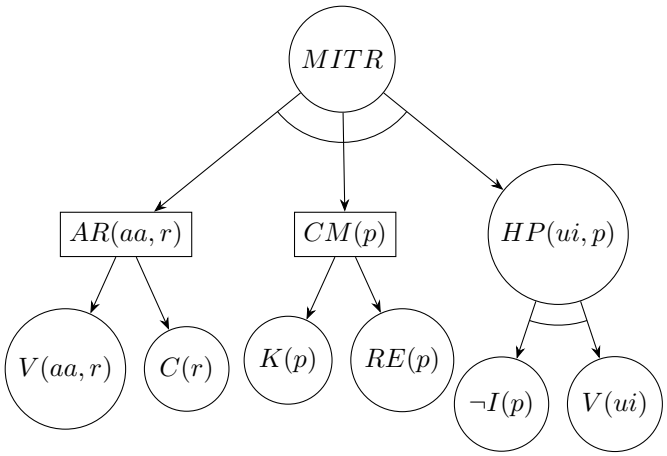

Figure 3: The generic model of MitR attack against any VR application/platform defined as a proof tree, tree notation based on [60]. The detailed description is provided in Equation (1).

- **Communication channel** for Command & Control (C&C) protocol to allow control and monitoring of infected zombies in a botnet.

To perform a MitR attack in a VR application, the attacker has the following goals:

- **Access targeted room** by exploiting vulnerabilities in authentication & authorization mechanisms or by obtaining credentials.
- **Connect to multimedia protocol** using RE techniques.
- **Hide presence** from User Interface (UI) and VE by exploiting lack of validation and verification in the protocol and vulnerabilities in application's UI.

When a VR application includes insecure code implemented in other software, it inherits these security risks. For example, in some cases, it could be possible to employ knowledge of attacking techniques used against network-enabled multimedia streaming applications relying on signaling channels over the Secure WebSockets (WSS) protocol, amongst other things demonstrated in Section 4.

We further detail MitR, it's goals, and requirements formally in Figure 3 and associated Equation (1). The successful realization of the attack against virtual room r of targeted VR application/platform, using communication & multimedia sharing protocol p, authentication & authorization mechanism aa, and the information is presented to users 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 VR 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 Section 4.8.

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

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 y with authentication & authorization mechanism x
$CM(x)$	$=$ 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

3. Threat/Adversary Model

The underlying hypothesis of the proposed model is that the attacker 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 (Figure 4). For testing purposes, we include the assumption that the target is also VR capable and has or will launch the application.

4. Case Study – Attacking Bigscreen

We provide a case study that shows a concrete demonstration of our MitR attack and VR Worm carried out against the Bigscreen application. The analyzed technologies are widely employed in other VR applications and platforms (e.g. Unity engine, C#, .NET, UI layer using JS & jQuery, WebRTC, HTTP, WebSockets (WS), Transport Layer Security (TLS)). It is possible to generalize evaluation of the Bigscreen to VR systems broadly. Our security analysis included methods of vulnerability research and penetration testing, it can be categorized as: 1. *External*, 2. *Black box*, 3. *Non-destructive*, 4. *Ethical*, and 5. *In a controlled environment*. The security analysis is broken down into several phases, which could be mapped to corresponding stages defined by [39] with some adjustments. The adjusted security phases we propose are characterized as follows:

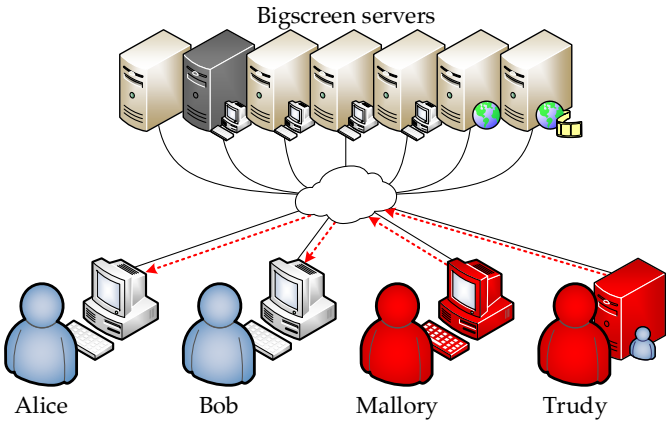

Figure 4: Basic scenario for attacking the Bigscreen application. Alice and Bob are legitimate users of the application, each in a different location. Mallory is an attacker with maliciously patched Bigscreen application (Figure 9). Trudy is an attacker with developed C&C server capable of attacking Bigscreen users and controling created botnet (Section 4.6). Mallory and Trudy aim at users of the application and do not attack Bigscreen servers (Figure 6, Figure 10).

- **Phase I – Reconnaissance** Examine the Bigscreen and gather publicly available information (detailed in Section 4.2).
- **Phase II – Laboratory Setup & Tool Sets** Prepare laboratory equipment and software tools based on identified areas of interest (described in Section 4.3).
- **Phase III – Security Analysis** Assess overall security with network traffic analysis, penetration testing, RE of used protocols and the Bigscreen desktop application (presented in Section 4.4).
- **Phase IV – Exploit Development** With identified security flaws, craft exploits to assess the impact of vulnerabilities (highlighted in Section 4.5).
- **Phase V – Tool Construction** Aggregate discovered attacks and exploits into a comprehensive attacking tool (outlined in Section 4.6).
- **Phase VI – Testing** Evaluate the success rate according to defined scenarios (Section 4.7). Summarize our results (Section 4.8) and responsibly disclose findings to vendors (Section 4.9).

4.1. Scenarios

We defined multiple scenarios for maintaining a systematic approach during analysis and testing. Scenarios refer to actions of hypothetical legitimate users (Alice, Bob) and attackers (Mallory, Trudy), as in Figure 4. Individual steps are available in *supplemental materials* as Appendix A. These scenarios cover the common functionalities offered by the Bigscreen application: 1. Passive stay in lobby, 2. Creation of a public room, 3. Creation of a private room, 4. Conducting a private meeting, and 5. Transitioning between rooms.

4.2. Phase I – Reconnaissance

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]

Our previous forensic research [25] identified several artifacts left on the hard drive by the Bigscreen application. We uncovered that the application's UI elements were controlled by JS (with jQuery) in a limited built-in web browser and the UI had bindings to the application's core layer.

4.3. Phase II – Laboratory Setup & Tool Sets

The testing and analysis described in the study was carried out in a controlled laboratory environment similar to our previous work [25]. The tools and equipment useful for the analysis are listed respectively in Table 1 and Table 2. The network topology used for experimentation purposes is depicted in Figure 5.

4.4. Phase III – Security Analysis

Most of the traffic generated by the Bigscreen 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. This technique allowed us to decrypt traffic protected by TLS.

We managed to RE and decompile the application and Dynamic Link Libraries (DLLs) into corresponding logic in C#, which revealed the inner structure of the application. Figure 7 presents a diagram of the developed exploit to download and execute malware on victim's computer. In fact, 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 Cross-site Scripting (XSS) attack via unsafe jQuery methods (Section 5.4). The attack propagates through

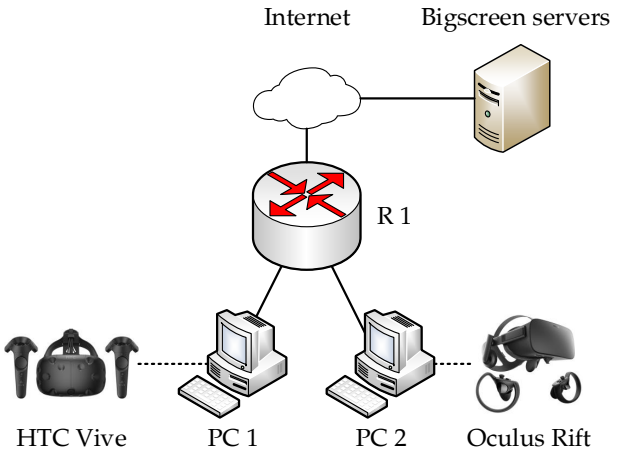

Figure 5: Initial Experimental Setup

bindings from UI to core layer. Application core then calls method from Unity with malicious payload which causes Remote Code Execution (RCE). Finally, attack escapes from the application, downloads malware and executes it.

We found out that DLLs are loaded without integrity checks. This made unauthorized patching of the Bigscreen application possible (i.e. Application Crippling).

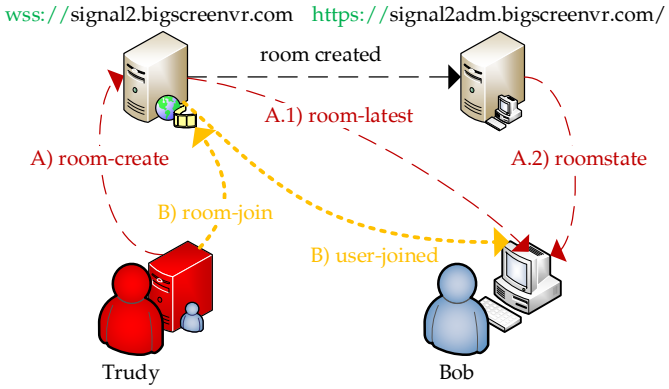

Figure 6: The two identified paths of forged signaling messages in an attack over the network. See Section 4.4 for details. *Supplemental materials* present example payloads in Appendix H and network traffic analysis in Appendix C.

Application's desktop UI was implemented as JS environment which communicates with C# core layer via JS-C# function bindings. This UI layer also communicates with Bigscreen's servers using WSS. Extracted JS source code was obfuscated and minified, but we managed to implement a deobfuscator for this format (Section 5.5). During analysis of JS-C# bindings, we discovered a critical security vulnerability in the C# Unity scripting Application Programming Interface (API). A method `Application.OpenURL(url)` is dangerously capable of running programs, opening folders and files on the host computer (see Sections 5.1 and 5.2 and Listing 12 and 17 in *supplemental*

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

Table 1: Tools for Security Analysis

Purpose	Software
network traffic analysis	mitmproxy, wifimitm, Scapy, Wireshark, Netfox Detective, NetworkMiner, *tools from [39, 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, JsArrayRefDeobfuscator (jsard)

Table 2: Software Versions of Virtual Reality Applications and Devices

Application / Device	Software Version
Bigscreen	0.34.0
Oculus App	1.36.0.215623
Steam	1549129917
SteamVR	1.2.10
Vive Headset MV HTC	1462663157
Vive Base HTC V2-XD/XE (x2)	436
Vive Controller MV HTC (x2)	1533720215
Rift Headset	709/b1ae4f61ae
Rift Sensor (x2)	178/e9c7e0406-4ed1bd7a089
Rift Touch (x2)	f3c65f7a5f

materials).

RE led to an understanding of the signaling protocol which was used to manage VR rooms and establish multimedia P2P channels. WSS channels were observed to use encryption but were fundamentally flawed due to a lack of authentication & authorization. We discovered it is possible to send a specially crafted message to the signaling server which is forwarded to either the room's participants or the application's lobby. At the network message level, the attacker can set arbitrary values to some fields. The application did not perform proper sanitization of data received through encrypted signaling channel from the signaling server. The application naively trusts its signaling server, which can forward untrusted messages. The received malicious payloads are propagated to the UI layer resulting in remote XSS over signaling channel. Attacking the lobby can affect all users of the Bigscreen application worldwide.

Figure 6 presents the two paths we identified to forge

Table 3: 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
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	Signaling channel	High
Lack of authentication, connection from a custom application	WebRTC	Medium
Information leak via RE of assemblies	Application core	Medium
Information leak via RE of obfuscated and minified JS source code	UI layer	Low

signaling messages in an attack over the network. 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 his application (`room-latest`), this is applicable to all users in the lobby. 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`).

Selected payloads of Bigscreen XSS attacks are available in *supplemental materials* as Appendix H. Please see [55] for a complete procedure, other discovered vulnerabilities, and technical details.

4.5. Phase IV – Exploit Development

In order to know the severity of the identified issues, we focused on ways how malicious hackers could possibly abuse the Bigscreen application and put its users at risk. The following highlights significant achievements while the full set of exploits and technical details are available in [55].

- **Eavesdropping victim's screen and microphone:** Forged signaling message can override victim's multimedia sharing over the network, as depicted in Figure 6. Our PoC WebRTC application was able to covertly connect to legitimate Bigscreen applications without the user's knowledge.

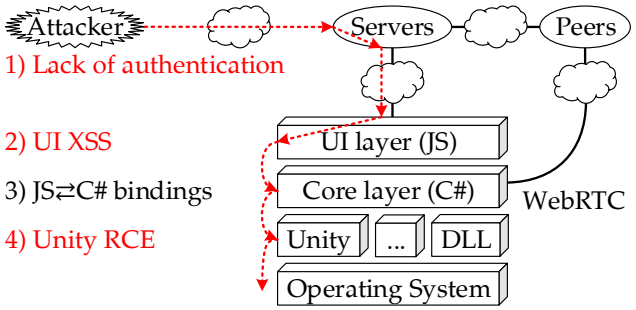

Figure 7: Diagram of developed exploit to download and execute malware on Bigscreen victim's computer. See Appendix H in *supplemental materials* for example payloads.

- **Discovery of private rooms:** XSS payload in a spoofed signaling message can force the victim to leak the private room ID to the attacker's C&C server.
- **Download & execute malware on victim's computer:** Security flaw in Unity scripting API can be exploited to remotely run programs, open folders and files. The attack downloads and executes malware, as illustrated in Figure 7.
- **Botnet and VR worm spreading through the whole Bigscreen community:** Combination of several discovered vulnerabilities allowed realization of VR worm, implementation is described in Section 4.8.
- **Man-in-the-Room attack:** A new cyber attack related to VEs has been successfully realized during this research (see details in Section 4.8).

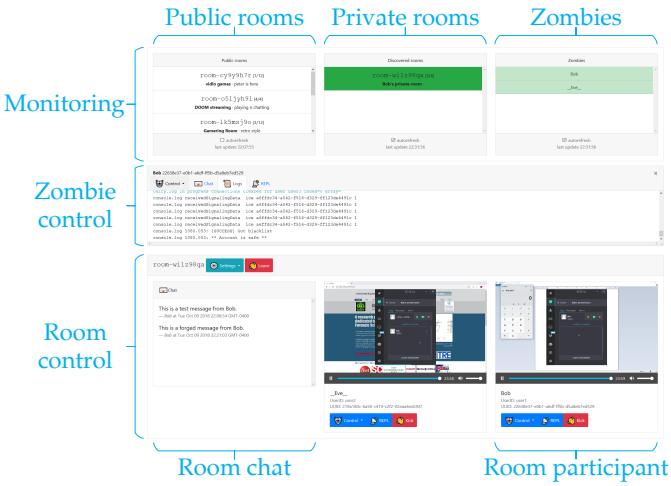

Figure 8: Visualization of the main parts of the C&C tool. *Zombie control* and *Room control* can be opened and closed for each controlled zombie and room. Each room can have multiple *Room participants*.

4.6. Phase V – Tool Construction

To understandably demonstrate our findings to the scientific community, we designed and implemented a user friendly attacking tool. It can execute individual attacks and serve as C&C (bot herder), controlling the entire botnet of infected Bigscreen applications. It also includes our PoC VR worm. Malware infection can be done with a single button.

The attacker can target any public or discovered private room and take control over the Bigscreen application. Dashboard can control multiple rooms at the same time using their *Room control*. Figure 8 highlights our C&C tool where *Zombie control* and *Room control* can be opened and closed for each controlled zombie and room. Chat messages are shown in *Room chat* panel. The attacker can eavesdrop victim's screen and microphone using *Room participant* panel. The tool monitors zombies and each of them can be controlled with corresponding *Zombie control*. We have developed our custom C&C protocol to control zombies from the C&C server.

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, C&C relay server, and payloads to execute all exploits (Section 4.5) is available online for research purposes.[3].

4.7. Phase VI – Testing

Testing was carried out according to defined scenarios (Section 4.1). 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:

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

All **tested attacks require no action from the victim (zero-click attacks)**. Individual detailed steps how all test cases were evaluated are in *supplemental materials* as Appendix B and summarized in Table 4.

Table 4: 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

[3] `https://github.com/unhcfreg/VR-MitR-C2-Bigscreen`

4.8. Major Findings

We conducted a security and forensic analysis including methods of traffic analysis, vulnerability research, penetration testing, and RE of the application and its network protocols. In this section, we summarized the main discovered vulnerabilities of the Bigscreen application (see Table 3). Opportunities in discovered vulnerabilities were then seized to create practical exploits highlighted in Table 6. Advanced attacks require chaining multiple exploits (Table 5). Details are available in [55].

We now explain how requirements of general attack concepts (Section 2.2) 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.

Table 5: Exploits Based on Combination of Vulnerabilities

Attack	UI XSS	Unity RCE	Patch	Severity
Man-in-the-Room.	✓	×	✓	High
VR Worm.	✓	×	×	High
Download & run malware.	✓	✓	×	High

Table 6: Selection of Exploits Based on Forged Signaling Messages with XSS Payloads

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 screen and microphone.	High
Privacy violation	Persistently eavesdrop victim's chat, even if they go to another room.	High
Impersonation	Spoof chat messages.	Medium
Privilege escalation	Set selected user as room admin.	Medium
DoS	Ban selected victim until restart.	Low

Listing 1: Minimal and simplified 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();
```

Requirements for a successful outbreak of VR worms and their botnets in Bigscreen were fulfilled as follows:

- ✓ **Vulnerable persistent environment:** Modifications by XSS attack can persist until application reset. Victims can, therefore, propagate the payload further.
- ✓ **Functionality for duplication of a worm:** An attacker can 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 10). The principle of the worm is illustrated in Listing 1.
- ✓ **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. 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.

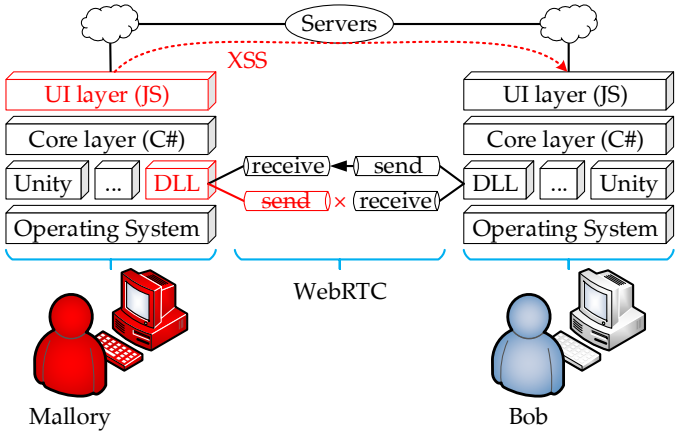

Figure 9: The attacker Mallory uses patched (application crippling) version of the application which does not send VR state to other room participants and which also uses forged signaling messages with XSS payloads to hide traces of Mallory's presence in the room.

The MitR attack goals associated with the Bigscreen case study were achieved as follows:

- ✓ **Access targeted room:** Remote XSS through signaling channel leaked confidential Room IDs of private rooms. Credentials for public rooms were publicly available.
- ✓ **Connect to multimedia protocol:** We patched some DLLs loaded by Bigscreen application (no integrity checking, Table 3) to change selected behavior. Our PoC patched Bigscreen application could connect with legitimate Bigscreen applications. This also gave us complete control over one end of audio/video/data streams (Figure 9).

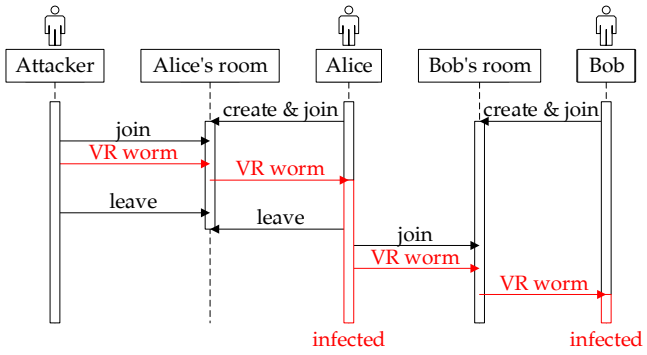

Figure 10: Sequence diagram of of initial VR worm infection (in Alice's room) and propagation from one user to another when they meet in VR room (in Bob's room).

✓ **Hide presence:** XSS payloads altered victim's UI. Combination of vulnerabilities (Figure 9, Table 5) allowed to bypass sharing of attacker's VR state. Victim had no data to render – the attacker was invisible in VE. The attacker could see victims in VR, see screens of their computers, hear their audio/microphone (Figure 11).

We can now utilize formal definition from Section 2.2 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:

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

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

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

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

4.9. Mitigations & Suggestions

We provided related vendors with suggested mitigation strategies. Both companies used these measures to remedy the issues [61, 62, 63, 64]. However, our mitigation strategy may be applicable to other VR vendors to improve their security posture. Responsible disclosure letters are in Appendix K and Appendix L in *supplemental materials*.

In the case of the Bigscreen application, discovered weaknesses were caused by shortcomings & vulnerabilities in authentication, authorization, encryption, data sanitization, integrity checking, or by a critical security vulnerability in the integrated 3rd party software (Unity engine). Individual flaws with smaller impact were chained together resulting in attacks with critical impact. As for the Unity Scripting API, discovered security vulnerability was based on unrestricted, undocumented, dangerously powerful, and unintended behavior of API method.

Furthermore, developers can detect and prevent such vulnerabilities even in any other software by adopting our FOSS tools (Section 5). For more details about specific mitigations and suggestions, please see Appendix I in enclosed *supplemental materials*.

5. Improving the State-of-the-Art of VR Vulnerability Detection & Prevention

We implemented a series of analytical tools and vulnerability signatures (Table 7). We opted to publish them as FOSS to improve to the state-of-the-art of vulnerability detection & prevention in VR. MitR attack exploits vulnerabilities that could have been prevented. Developers and scientists can now use our tools to make secure VR software.

[5] `https://youtu.be/N_Z3mfzLZME`

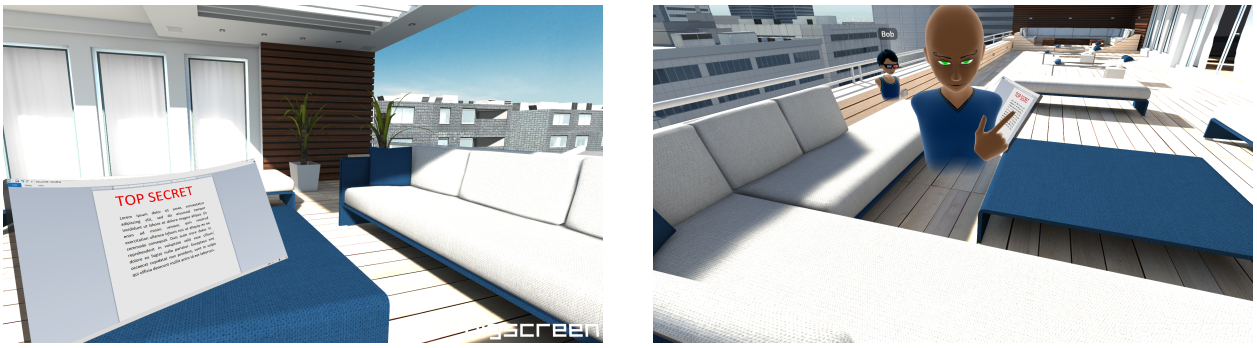

Figure 11: Our novel Man-in-the-Room attack. Figure on the left shows the view of the user Bob, figure on the right shows attacker Mallory who is invisible while in the room with Bob. The attacker uses malicious version of the application, as shown in Figure 9. Details are available in our video demonstration[5].

Table 7: Overview of our FOSS contribution to state of the art of automated vulnerability detection & prevention.

Project	**Goal**
CodeQL Signature for Unity OpenURL Vulnerability	Detect `OpenURL` RCE vulnerability in any other software built with Unity engine.
Unity OpenURL Exploit Demo	Raise VR developers' awareness about `OpenURL` RCE vulnerability.
jQuery XSS Static Analyzer	Detect XSS in UI layer or in any other web applications.
Dataset of Unsafe jQuery Method Calls	Evaluate our XSS analyzer and compare it with ESLint.
JavaScript Array Ref Deobfuscator	Revert *Array Ref* obfuscation format of JS.
SlimIt Library Fork	Improve JS parsing features for (not only) our XSS analyzer and deobfuscator.

5.1. CodeQL Signature for Unity OpenURL Vulnerability

CodeQL allows for powerful Static Application Security Testing (SAST) including taint analysis and data flow analysis. For example, GitHub is using it for batch scanning for vulnerabilities. We defined a CodeQL query and a build configuration which can detect vulnerabilities in Unity's `OpenURL` method. This is the vulnerability that we discovered and responsibly disclosed, see Appendix K and Appendix L in *supplemental materials*. The vulnerability signature is publicly available at the repository[6].

However, CodeQL database creation is currently available only as a per-request service on LGTM[7]. In the case of C# projects, CodeQL supports Microsoft Visual Studio builds with msbuild, while Unity-based projects can be compiled 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.

5.2. Unity OpenURL Exploit Demo

We implemented an example vulnerable Unity application to fully demonstrate all the possible cases how `OpenURL` can be exploited. Our aim is to raise developers' awareness about this vulnerability. The demo exploit application is available from its repository[8].

Example of exploiting `openLink` function inside Bigscreen's JS UI, which subsequently calls `Application.OpenURL` method from Unity engine is available in *supplemental materials* as Listing 12 and 17.

5.3. Dataset of Unsafe jQuery Method Calls

To evaluate our analyzer (Section 5.4) and to compare it with other tools, we created a dataset[9]. It consists of nearly 400 different samples of code and focuses on unsafe jQuery method calls. The dataset was created as a combination of static code samples and samples generated from templates created for each analyzed method. "By design, any jQuery constructor or method that accepts an HTML string – `jQuery()`, `.append()`, `.after()`, etc. – can potentially execute code. This can occur by injection of script tags or use of HTML attributes that execute code." ([65]) Samples are organized into two groups:

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

[6] `https://github.com/mvondracek/Unity-OpenURL-Exploit-Demo/tree/main/CodeQL`

[7] `https://lgtm.com/`

[8] `https://github.com/mvondracek/Unity-OpenURL-Exploit-Demo`

[9] `https://github.com/mvondracek/jQuery-XSS/tree/master/dataset`

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 the ability to perform taint analysis and data flow analysis is required. This means the tool should determine how malicious data are sanitized on paths from *sources* to vulnerable *sinks*. It should also describe how malicious data are propagated through expressions across these *sources* and *sinks* .

5.4. jQuery XSS Static Analyzer

The XSS in Bigscreen's UI layer was caused by unsafe Hypertext Markup Language (HTML) manipulation with jQuery methods (Section 4.4). HTML should be manipulated in a safe way using alternative approaches to avoid such software development weaknesses.

To prevent such vulnerabilities, we implemented a static analyzer for JS which can detect use of unsafe jQuery methods which are vulnerable to XSS attack. This analyzer is available as a Command-line Interface (CLI) program, but also as a plugin for Coala static analysis system[10]. Plugins for Coala are called bears[11] and this jQuery XSS Static Analyzer is released as *JSjQueryXssUnsafeBear*. For example, it can be used as part of SAST stage of Continuous Integration (CI) by developers to make their software safer. The analyzer is a FOSS and available from its repository[12]. Its algorithm is presented in Figure 12.

To evaluate our tool, we use the above-described dataset (Section 5.3). We also use this dataset to compare our tool with ESLint[13] and also ESLint with *jquery-unsafe* plugin[14]. The results we obtained from the evaluation of the involved tools are illustrated in Table 8, Table 9, and Table 10.

Table 8: 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%
		0%	0%		0%	0%	

We can see that ESLint lacks the ability to detect jQuery unsafe calls (Table 8). ESLint with a plugin cannot perform taint analysis or data flow analysis (Table 9). 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 also because its internal list of unsafe methods is incomplete. On the other hand, we can see some false positives (Table 9), where it incorrectly detected method calls not related to jQuery.

Table 9: Confusion matrix for evaluation of eslint tool with jquery-unsafe plugin with the dataset.

		Context Awareness		
		P	N	
eslint + jquery-unsafe	P	21	7	75.00%
	N	51	284	15.22%
		29.17%	2.41%	

		Taint & Data Flow		
		P	N	
eslint + jquery-unsafe	P	0	7	0%
	N	12	5	70.50%
		0%	58%	

Table 10: Confusion matrix for evaluation of our jQuery XSS Static Analyzer with the dataset.

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

Our analyzer utilizes power of a context-aware parser and detailed knowledge of unsafe jQuery methods (Figure 12). This way we managed to eliminate false positives and false negatives in the context-aware part of the dataset (Table 10). 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 the required techniques. The obtained results show ESLint with a plugin performs better than our tool (Table 9). In fact ESLint with a plugin is not aware of all unsafe methods, therefore it flags unknown ones as safe.

Our tool reaches 100% True Positive Rate (TPR), 0% False Positive Rate (FPR), 100% Positive Predictive Value (PPV), 0% False Omission Rate (FOR), for part of the dataset focused on context awareness (Figure 12). The dataset, test scripts, and results of the evaluation are publicly available[15].

[10] `https://coala.io/`
[11] `https://github.com/coala/coala-bears`
[12] `https://github.com/mvondracek/jQuery-XSS`
[13] `https://eslint.org/`
[14] `https://github.com/cdd/eslint-plugin-jquery-unsafe`
[15] `https://github.com/mvondracek/jQuery-XSS/tree/master/dataset`

```
Require: source = Javascript source code
Ensure: detections = vulnerable jQuery method calls
vulnerable = static list of vulnerable methods
detections ← [ ]
tree ← parse(source) /* abstract syntax tree */
for all node in tree do /* depth-first search (DFS) */
    if node is a method call and node has arguments then
    /* i.e. a possible setter method */
        access ← node.identifier
        if access.node is a jQuery selector expression then
        /* e.g. $("#selector") */
            if (access is a dot access and access.identifier.value is
            in vulnerable) then /* e.g. .html */
                insert node into detections
                /* e.g. $("#selector").html(...) */
            else if (access is a bracket access and access.expr is in
            vulnerable) then /* e.g. ["html"] */
                insert node into detections
                /* e.g. $("#selector")["html"](...) */
            end if
        end if
    end if
end for
return detections
```

Figure 12: Algorithm of our analyzer to detect vulnerable jQuery method calls in JS source code. Please note that conditions are separated here for a better readability.

5.5. 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[16].

5.6. SlimIt Library Fork

The above mentioned deobfuscator and static analyzer 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 SlimIt library[17]. It primarily offers a JS minifier, but also serves as a library with a JS parser. Unfortunately, SlimIt parser was missing some features that our tools needed. So we decided to create our fork[18] 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.

[16] https://github.com/mvondracek/JsArrayRefDeobfuscator
[17] https://slimit.readthedocs.io/en/latest/
[18] https://github.com/mvondracek/slimit

Table 11: Userbase

Software	Reach
Bigscreen	over 500,000 users[19]
Unity	3,000,000,000 devices[20]

6. 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 privacy and security risks. In addition, the products they bring to market are often released while still under development. We reacted to this by bringing our research to public attention in global media [55, p. 83]. We managed to publish the results together with explanation and recommendations for common users and the scientific and professional communities 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 (Appendix K in *supplemental materials*) and implemented appropriate security measures so that the described attacks may be mitigated [61]. Furthermore, Unity Technologies company addressed their issue by updating documentation of the dangerous method, as we suggested. After our responsible disclosure in 2018 (Appendix L in *supplemental materials*), several warnings have been included, so developers using the `OpenURL` method are now aware of its power and are instructed how to utilize it safely [63, 64]. Later in 2019, Unity's security team published a security warning for developers about the issue we reported [62]. "However, if the game developer does not properly sanitize what is passed into Application.OpenURL, their player could be at risk. [...] the victim's machine will immediately run the application at that link, potentially allowing an attacker to take control of the victim's system." ([62])

[19] https://bigscreenvr.com/press/
[20] https://unity3d.com/public-relations

We helped detect and prevent vulnerabilities which were discovered and exploited during this research (responsible disclosure letters are in Appendix K and Appendix L in *supplemental materials*). We released several analytical and attacking tools, example exploits, evaluation dataset, and vulnerability signatures so that scientific and professional communities ensure secure VR software development. Our work has impacted practice, and a popular VR application and development platform was improved significantly. The broad userbase of affected software is presented in Table 11.

We delivered a concept of novel VR attacks and we further identified key requirements for their realization 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.

7. Future Work

Our attacks were demonstrated on a single application. Future work aims to explore the automation of our approach so that it expands to other existing and future VR systems. The various analytic tools and vulnerability signatures we published can be easily utilized by the scientific and professional communities. Future work will also focus on both legal and policy implications of our findings as VR technology gains more momentum.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1748950. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The work was also supported by the Brno University of Technology internal project "Application of AI methods to cyber security and control systems" FIT-S-20-6293.

References

[1] M. Larin, "Overview of current trends in the field of virtual reality," *Common Information about the Journal A&SE*, p. 15, 2021.

[2] M. Mekni, *Automated generation of geometrically-precise and semantically-informed virtual geographic environments populated with spatially-reasoning agents.* Universal-Publishers, 2010.

[3] Y. Zhang, H. Liu, S.-C. Kang, and M. Al-Hussein, "Virtual reality applications for the built environment: Research trends and opportunities," *Automation in Construction*, vol. 118, p. 103311, 2020.

[4] I. E. Sutherland, "A head-mounted three dimensional display," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, ser. AFIPS '68 (Fall, part I). New York, NY, USA: ACM, 1968, pp. 757–764. [Online]. Available: http://doi.acm.org/10.1145/1476589.1476686

[5] J. Lanier, *Dawn of the New Everything: Encounters with Reality and Virtual Reality.* Henry Holt and Company, 2017. [Online]. Available: https://books.google.cz/books?id=8MCuDgAAQBAJ

[6] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel, "Reality built for two: A virtual reality tool," *SIGGRAPH Comput. Graph.*, vol. 24, no. 2, p. 35–36, feb 1990. [Online]. Available: https://doi.org/10.1145/91394.91409

[7] ——, "Reality built for two: A virtual reality tool," in *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, ser. I3D '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 35–36. [Online]. Available: https://doi.org/10.1145/91385.91409

[8] N. Sala, "Virtual reality, augmented reality, and mixed reality in education: A brief overview," *Current and prospective applications of virtual reality in higher education*, pp. 48–73, 2021.

[9] Oculus VR. (2015, May) First Look at the Rift, Shipping Q1 2016. Accessed: 2022-08-13. [Online]. Available: https://www.oculus.com/blog/first-look-at-the-rift-shipping-q1-2016/

[10] T. Merel. The reality of VR/AR growth. Accessed: 2021-10-01. [Online]. Available: https://techcrunch.com/2017/01/11/the-reality-of-vrar-growth/

[11] Steam. Accessed: 2021-10-01. [Online]. Available: https://store.steampowered.com/

[12] D. Heaney. (2019, Jan.) Share of VR headsets on Steam doubled in 2018. Upload VR. Accessed: 2021-05-19. [Online]. Available: https://uploadvr.com/vr-steam-grew-2018/

[13] B. Lang. (2021, Jan.) Monthly active VR headsets on Steam pass 2 million milestone. Road to VR. Accessed: 2021-05-19. [Online]. Available: https://www.roadtovr.com/steam-survey-vr-monthly-active-user-2-million-milestone/

[14] D. Heaney. (2021, Feb.) More than 2% of Steam users now have a VR headset. Upload VR. Accessed: 2021-05-19. [Online]. Available: https://uploadvr.com/steam-vr-users-2-percent/

[15] Steamworks Development. (2021, Jan.) 2020 year in review - Steam news. Steam. Accessed: 2021-05-19. [Online]. Available: https://store.steampowered.com/news/group/4145017/view/2961646623386540826

[16] ——. (2022, Mar.) 2021 year in review - Steam news. Steam. Accessed: 2022-08-08. [Online]. Available: https://store.steampowered.com/news/group/4145017/view/3133946090937137590

[17] J. Koetsier. (2018, Apr.) VR needs more social: 77% of virtual reality users want more social engagement. Forbes. Accessed: 2021-05-19. [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/

[18] Facebook Technologies, LLC. (2020, Oct.) Facebook Horizon - virtual reality worlds and communities. Oculus. Accessed: 2021-05-19. [Online]. Available: https://www.oculus.com/facebook-horizon/

[19] Microsoft. AltspaceVR. Accessed: 2021-05-19. [Online]. Available: https://altvr.com/

[20] vTime Holdings Limited. vTime - reality reimagined. Accessed: 2021-05-19. [Online]. Available: https://vtime.net/

[21] VRChat Inc. VRChat. Accessed: 2021-05-19. [Online]. Available: https://hello.vrchat.com/

[22] Bigscreen, Inc. (2018, Oct.) Press kit. Bigscreen. Accessed: 2019-01-14. [Online]. Available: https://bigscreenvr.com/press/

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] P. Casey, R. Lindsay-Decusati, I. Baggili, and F. Breitinger, "Inception: Virtual space in memory space in real space – memory forensics of immersive virtual reality with the HTC Vive," *Digital Investigation*, 7 2019.

[28] University of New Haven. (2019, Feb.) University of New Haven Researchers Discover Critical Vulnerabilities in Popular Virtual Reality Application. Accessed: 2022-03-20. [Online]. Available: https://www.newhaven.edu/news/releases/2019/discover-vulnerabilities-virtual-reality-app.php

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] J. Happa, M. Glencross, and A. Steed, "Cyber security threats and challenges in collaborative mixed-reality," *Frontiers in ICT*, vol. 6, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fict.2019.00005

[35] J. A. De Guzman, K. Thilakarathna, and A. Seneviratne, "Security and privacy approaches in mixed reality: A literature survey," *ACM Comput. Surv.*, vol. 52, no. 6, oct 2019. [Online]. Available: https://doi.org/10.1145/3359626

[36] S. Rokhsaritalemi, A. Sadeghi-Niaraki, and S.-M. Choi, "A review on mixed reality: Current trends, challenges and prospects," *Applied Sciences*, vol. 10, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/2/636

[37] D. Adams, A. Bah, C. Barwulor, N. Musaby, K. Pitkin, and E. M. Redmiles, "Ethics emerging: the story of privacy and security perceptions in virtual reality," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 427–442. [Online]. Available: https://www.usenix.org/conference/soups2018/presentation/adams

[38] 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

[39] 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.

[40] 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.

[41] 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.

[42] 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

[43] 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.

[44] 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

[45] 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

[46] 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.

[47] M. Vondráček, J. Pluskal, and O. Ryšavý, "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

[48] 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/

[49] 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.

[50] 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

[51] 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

[52] 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

[53] 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.

[54] N. Montfort, "Obfuscated code," *Software Studies: A Lexicon*, pp. 193–199, 2008.

[55] M. Vondráček, "Security Analysis of Immersive Virtual Reality and Its Implications," Master's thesis, Brno University of Technology, Faculty of Information Technology, 2019.

[56] W. H. Securtiy, "Cross site scripting worms and viruses, the impending threat and the best defense," 2006.

[57] 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.

[58] 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.

[59] V. B. Livshits and W. Cui, "Spectator: Detection and containment of javascript worms." in *USENIX Annual Technical Conference*, 2008, pp. 335–348.

[60] A. Kishimoto, M. Winands, M. Müller, and J.-T. Saito, "Game-tree search using proof numbers: The first twenty years," *ICGA journal*, vol. 35, pp. 131–156, 09 2012.

[61] D. Shankar. (2019, Feb.) The bigscreen beta "2019 update" is now live! Bigscreen Blog. Accessed: 2022-08-13. [Online].

Available: https://blog.bigscreenvr.com/the-bigscreen-beta-2019-update-is-now-live-a68ab6ceb506
[62] B. Caldwell. (2019, Nov.) How to use URL handlers and OpenURL safely in your Unity app. Unity Technologies. Accessed: 2022-08-28. [Online]. Available: https://blog.unity.com/technology/how-to-use-url-handlers-and-openurl-safely-in-your-unity-app
[63] Unity Technologies. (2019, Apr.) Unity - Scripting API: Application.OpenURL, v2018.3. Unity Technologies. Accessed: 2022-08-13. [Online]. Available: https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Application.OpenURL.html
[64] ——. (2019, Aug.) Unity - Scripting API: Application.OpenURL, v2019.1. Unity Technologies. Accessed: 2022-08-13. [Online]. Available: https://docs.unity3d.com/2019.1/Documentation/ScriptReference/Application.OpenURL.html
[65] OpenJS Foundation. jQuery API Documentation. Accessed: 2022-03-19. [Online]. Available: https://api.jquery.com/html/