Persistent Cross-site Scripting (XSS) vulnerability in SVG attachments in PrivateBin < v1.4.0

Moderate **elrido** published **GHSA-cqcc-mm6x-vmvw** on Apr 9

Package

php **privatebin** (Composer)

Affected versions	Patched versions
>= v0.21, < v1.4.0	v1.4.0

privatebin/gcs (Docker)

>= v0.21,< v1.4.0	v1.4.0

privatebin/nginx-fpm-alpine (Docker)

>= v0.21, < v1.4.0	v1.4.0

privatebin/pdo (Docker)

>= v0.21, < v1.4.0	v1.4.0

Description

In PrivateBin < v1.4.0 a cross-site scripting (XSS) vulnerability was found. The vulnerability is present since attachments with image preview got introduced in v0.21 of the project, which was at the time still called ZeroBin. The issue is caused by the fact that SVGs can contain JavaScript. This can allow an attacker to execute code, if the user opens a paste with a specifically crafted SVG attachment, and interacts with the preview image and the instance isn't protected by an appropriate content security policy.

As a consequence, we have mitigated the vulnerability in the preview and urge server administrators to either **upgrade** to a version with the fix or to ensure the content security policy of their instance is set correctly, ideally both. Additionally, we expanded our directory listing tool with a checking mechanism and **highly suggest server administrators to check their instance there** and, should there be a warning regarding the content security policy **adjust the CSP to our suggested one**, as it is shown in the configuration preset.

Proof of concept

The vulnerability can be triggered as following:

1. Create the following SVG as a file:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
 <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
 <script type="text/javascript">
   alert(document.domain);
 </script>
</svg>
```

2. Upload it as an attachment to a PrivateBin instance that has attachments enabled and hasn't set the recommended content security policy (in particular, one that has either no content security policy set or that allows `*` or `blob:` as a `script-src`).
3. Open the paste. (In a real attack scenario this would be done by the victim.)
4. The SVG is rendered safely as a preview, and script isn't yet executed.
5. Now (depending on your device) right-click or long tap on the image and open it in a new tab.
6. Now a `blob:` URI opens in a new tab with the image and the modal is shown, therefore the script got executed.

Impact

We tried to reproduce the vulnerability and in our assessment, we found out the following:

1. Any users who use our recommended *Content Security Policy* (CSP), even older, less strict ones, are **not affected** by this vulnerability, if a CSP compatible browser is used. All the browsers we tested with did pass on the CSP to the new tab that is opened when viewing the SVG by itself.
   As PrivateBin ships with a built-in CSP, we consider this a strong defence in depth against these and related issues. That said, we think the CSP should only be the last layer of defence and as such, we decided to still apply further mitigations for this security issue.
2. Instances that do not have attachments enabled, are not affected. Even when attachments are uploaded using a third-party client, they can't be rendered when the administrator disables them

(the HTML element that they would render in isn't present and before 1.4 this caused an error, we now catch the error and only display the paste text) and thus potential exploits in the attachment file do not apply.

3. The inline preview (step 4 above) does *not* execute the script, because browsers explicitly restrict SVGs if the they are is embedded in an `img` tag to prevent such security issues in images. Thus, SVGs in `img` tags itself can be considered safe.
   However, when the user opens the SVG in a new tab, this browser security feature is circumvented. That's why the exploit steps above explain to open the SVG in a new tab. That being said, the impact of the vulnerability is reduced by two factors:
   i. The attack explicitly requires **user-interaction**, i.e. the user has to be tricked into opening the preview in a new tab for some reason. This could realistically be achieved with some social engineering: The markdown formatted text part of the paste could include such an instruction as a big, bold title, or the SVG could be very large and have very small text, which the user might want to zoom into, in order to read.
   ii. Potential exploit code can only run in a new tab. It still has the same origin (as can be seen in the PoC above, because the domain/origin the script is running on, is shown). However, though, sensitive information like the paste content, potential comments or encryption key (in the URL) is not accessible to the attacker as the context is now a blob-URL – and would anyway consist mostly of things the attacker initially created itself.
   That said, the same origin policy applies and thus, what an attacker could do is read e.g. cookies and local storage data saved in the same origin. As PrivateBin itself does not use any of that, the impact of this vulnerability is limited. However, as PrivateBin is a software for self-hosting, it cannot be excluded that other services run in the same origin (e.g. on the same domain). That's why server administrators may need to evaluate the impact of running arbitrary JavaScript code on their domain/origin where PrivateBin is hosted by themselves.

To summarize, this shows a fairly limited impact, given even if the CSP had not caught the issue, the user still needs to interact with the page and the exploit code cannot access or exfiltrate any data of the PrivateBin instance.
Note: However, take our assessment only as a basis for your own assessment. As explained, depending on your environment, the actual risk may vary if you are hosting other services on the same domain as PrivateBin.

As for the metrics, the impact assessment we have done with CVSS v3.1 results in this:
AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N/E:F/RL:U/RC:C/CR:X/IR:X/AR:X/MAV:N/MAC:L/MPR:N/MUI:R/MS:U/MC:X/MI:X/MA:X

Real-life impact

We found two affected instances in our instance lists (wiki, directory) that did not serve a correct Content Security Policy header, had attachments enabled and thus are vulnerable to this attack. We didn't manage to get into contact with the administrators of these sites, though.

In addition to that, we found that multiple instances do seem to either strip our CSP or have it changed to an unsafe setting and have thus expanded our directory service to verify whether our recommend CSP is used or not. (see below)
We have no reports that indicate this vulnerability was or is being actively exploited at the time of this report.

Patches

To fix the problem, we took the following measures (in no particular order):

- We apply DOMpurify (a library we already use to sanitize user-submitted HTML via the Markdown format) to the SVG preview, too. It strips script tags and other uncommon security-relevant and potentially malicious tags/properties from the SVG file.
  So whether you open the SVG in a new tab or not and whether CSP is present and enabled or not does not matter any more, as the displayed SVG is sanitized.
- As a further defence in depth mechanism we now send the CSP both as an HTTP header, as well as a meta tag. This protects instance with mis-configured web servers, CDNs, proxy or similar, from stripping or breaking the CSP headers, as they still get the CSP inside of the HTML content itself. Please note though, that the meta tag approach is not as strong as the HTTP header approach and should thus only be considered as a fallback.
- The PrivateBin Directory now also scans whether the recommend Content-Security-Policy header is used on a given instance. If you do not want to have your website appear in the list, but check it manually you can use a separate check page there.

The code-changes in PrivateBin can be found in #906.

Note: Please note that we explicitly chose to *not* apply *DOMPurify* if you download the (SVG) attachment with the download button. Subsequently, if a user would manually opens the downloaded SVG in the browser, it will be opened from the `file://` protocol and thus from a different origin, so all reference to the download location is lost and no more security risk is associated with that, than opening any website or local HTML file. Thus, the SVG file with stay intact in it's original form, if you download the attachment.
We consider the execution of code from attachments outside of the PrivateBin instance's context to be out of scope to mitigate (i.e. malware in executables, office documents macros, PDF scripts), as all of these require client side mitigations to be applied to all such downloaded file types, independent on where they get downloaded from.

Workarounds

We strongly recommend you to upgrade to our latest release, especially as we also upgraded outdated and potential vulnerable libraries (see below). However, here are two workarounds that may help you to mitigate this vulnerability:

- Update the CSP in your configuration file to the latest recommended settings and check that it isn't getting reverted or overwritten by your web server, reverse proxy or CDN, i.e. using our offered check service.

- Deploying PrivateBin on a separate domain may limit the scope of the vulnerability to PrivateBin itself and thus, as described in the “Impact” section, effectively prevent any damage by the vulnerability to other resources you are hosting.
- As explained in the impact assessment, disabling attachments also prevents this issue.

References

We highly encourage server administrators and others involved with the PrivateBin project to read-up on how Content-Security-Policies work, especially should you consider to manually adjust it:

- https://content-security-policy.com/
- https://developer.mozilla.org/docs/Web/HTTP/CSP
- https://developers.google.com/web/fundamentals/security/csp/

Also please note that if multiple headers are set (as e.g. done via our now introduced meta tag) browsers should apply the most restrictive set of the policies, as per the CSP specification.

For more information

Please notice we also upgraded jQuery that was reported to us as being vulnerable both by our automated container security scanning as well as by users.
By doing so, we also updated all other dependencies we use. Our tooling identified the following vulnerabilities in jQuery:

- CVE-2020-11023
- CVE-2020-11022

In a limited assessment about these when we were made aware of them we could not find any immediate risk, but nevertheless, we encourage users to upgrade to be on the safe side.

Finally, we also upgraded zlib to address CVE-2018-25032.

Timeline

- 2022-02-22 – Initial contact by reporter.
- 2022-02-25 – Reporter sends in a detailed report.
- 2022-02-26 – Report gets reviewed, initial findings around the content security get shared and reporter withdraws report.
- 2022-04-09 – Vulnerability details published.

Credits

This vulnerability was reported by Ian Budd, Nethemba s.r.o, which we'd like to thank for that.
In general, we'd like to thank everyone reporting issues and potential vulnerabilities to us.

If you think you have found a vulnerability or potential security risk, we'd kindly ask you to follow our security policy and report it to us. We then assess the report and will take the actions we deem necessary to address it.

Severity

Moderate 4.3 / 10

CVSS base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	Required
Scope	Unchanged
Confidentiality	Low
Integrity	None
Availability	None

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N

CVE ID

CVE-2022-24833

Weaknesses

CWE-79 CWE-80