Persistent XSS vulnerability in filename of attached file in PrivateBin < 1.3.2/1.2.2 (2020-01-11)

Moderate elrido published GHSA-8j72-p2wm-6738 on Jan 11, 2020

Package

privatebin (Composer)

Affected versions	Patched versions
>= 1.2	1.3.2 / 1.2.2

Description

On 24th of December 2019 one of the property based unit tests reported a failure. Upon investigation, @elrido discovered that the failure was due to unescaped HTML, which allowed the user provided attachment file name to inject HTML under certain conditions leading to a persistent Cross-site scripting (XSS) vulnerability. After committing an initial fix to the master branch, the issue was reported on 25th of December. Vulnerability write-up done by @rugk and @elrido.
The vulnerability has been fixed in PrivateBin v1.3.2 & v1.2.2. Admins are urged to upgrade to these versions to protect the affected users.

Affected versions

Any PrivateBin version since 1.2.

Conditions

- The configuration setting `fileupload` has to be enabled, as 1.3 displays an error when trying to open a paste with attachment.
- The CSP header rules don't get applied. For example:
  - They are unsupported or disabled in the visitors browser.
  - They are filtered out by a some proxy server at the server or client side.
  - The header is disabled/commented in the PHP-logic.
  - The rules have been relaxed in the `cspheader` configuration setting.
- A paste with a malicious filename is created.
- A visitor of that paste clicks on the "Clone" button.

Proof of concept

The following method is just one possibility to exploit this issue and is provided as a proof of concept with steps to reproduce the issue. To avoid having to create an actual file with a rogue filename, one could use the Python CLI client for PrivateBin and edit line 56 in `format.py` as follows:

```
-        self._attachment_name = path_leaf(path)
+        self._attachment_name = '<script>alert("☹️");//<a'
```

The paste then can be created on a vulnerable instance:

```
$ python pbincli/cli.py send -t " " -f /dev/null -s https://privatebin.net/
```

Visiting the created paste on a vulnerable instance, with `fileupload` enabled and the CSP header weakened or disabled, and clicking the clone button will insert the HTML unescaped. In the above example, a pop-up would appear, when the script is executed.

Impact

On a vulnerable site pastes with malicious filenames can be created and users visiting these could inadvertently trigger the code execution when they click the "Clone" button. They could be instigated to do so via social engineering, like a paste text suggesting to "clone and share" the paste in question.

The attached file itself doesn't have to be empty and could be an image or document that would still be displayed inline as usual. The execution of the script triggered by clicking on the "Clone" button may occur silently in the background without showing any noticeable signs in browser. It may for instance be used to track visitors, start drive-by-downloads or similar. While we focus on script injection here, as it is the easiest exploit vector, it has to be said that anything else can be injected like CSS, images, videos, although the default CSP will block inline CSS and images, e.g.

On first visit, the filename isn't visible and is properly escaped when inserted into the download attribute. Only when clicking the "Download attachment" link would open a file save dialog with an odd name pre-filled, although browsers will convert illegal characters into valid ones, so it may not be identical to the one provided. Only when the "Clone" button has been clicked and after the exploit has already been triggered, the filename gets displayed. Note that an attacker can of course prevent this indicator of compromise to be shown and e.g. change the displayed text or obfuscate the filename by starting it with something harmless, like `image.png`, before opening the HTML tag.

Impact restrictions

The impact is mitigated by the fact that the vulnerability is, as far as our investigation concluded, paste-specific, i.e. opening a vulnerable paste can only compromise this one paste and no other pastes.

Furthermore, as stated before, the impact is mitigated by the fact that we deploy a strong CSP that, by default, does not allow inline JS or other potentially easy methods that would allow an easy exploitation of the vulnerability.
That said, we have to make users aware, that there may always be tricks to bypass such a CSP and the simple injection of HTML tags, e.g. destroying, faking or somehow phishing an HTML page does always remain a possibility.

As such, we treat this as a security vulnerability with medium severity. It is critical on it's own, but we believe that the deployed security mechanisms should prevent an exploit in practice in most cases.

Real-life impact

We checked all instances listed in the PrivateBin directory in the Wiki and didn't find any 1.1, 1.2 or 1.3 instances that had the CSP headers disabled or in a state we know to be vulnerable. We used the following script, that may be adapted to check the CSP headers of any single instance:

```
for URL in $(
    curl -s https://raw.githubusercontent.com/wiki/PrivateBin/PrivateBin/PrivateBin-Directory.md | grep -Po '^http.*?(?= )'
)
do
    echo -n "$URL: "
    curl -sLI $URL | grep -Poi 'content-security-policy.*script-src.?\K.*?(?=;)' || echo 'No CSP detected!'
done
```

Some of the above sites do offer file uploads. On these instances, it is still possible that a visitor could have CSP support disabled in their browser, i.e. via a transparent proxy at their internet uplink or due to a browser plugin or some other locally installed, misguided security solution.

Important: This scan is only a cursory check and *must not* be taken as a security analysis of any means! You are always responsible for the security of your own instance. Please do check it manually!

Mitigation

As server administrators can't detect if a paste contains file attachments at all (apart from their size) in version 1.3 and due to the encrypted filename in older versions, as well as the risk for clients that don't apply the CSP settings, we urge them to upgrade to versions 1.3.2 or 1.2.2.

If you use v1.3, you could disable the `fileupload` setting to prevent pastes from getting displayed that may contain this vulnerability. Note that this will break all existing pastes with uploads, however, so we do *not* recommend this, but advise you to upgrade to a fixed version instead.

Further information

We want to make potential third-party client authors aware of this vulnerability and urge them to double-check their implementations. If they develop a client that displays untrusted foreign data from a paste in a HTML site, please make sure to escape it to prevent XSS attacks. Such attacks can only be prevented when the paste is displayed, a mitigation when a paste is created is pointless, as a different client can be used during creation.

We do also acknowledge and want to highlight the benefit of the CSP, which has first been introduced in PrivateBin v1.0.

However, we want to make you again aware that a whitelist-based CSP as we use, may sometimes still be bypassed, e.g. if you host a copy of the Angular library on the same domain as your PrivateBin instance.
We are aware of that and do consider upgrading to a stronger CSP in the future.

Issue discovery

While it is satisfying that our hard work on introducing unit tests has payed off in the discovery and mitigation of this vulnerability, it does also show a limitation of unit testing. A third party doing a code review would have certainly focused on how the project handles user provided input and may have discovered this much quicker.

The discovery wasn't due to the unit test checking for HTML input to get properly escaped, on the contrary, the test assumed input would not be changed. So other instances of HTML tags generated would have happily passed the test. Only when the test generated a fragment of a link (`<a`) it failed, because the DOM silently removed it when it inserted the string, as links within links aren't allowed. While the test was written to throw arbitrary strings at the `AttachmentViewer.moveAttachmentTo()` method, the test would only check that these got inserted into the DOM unchanged, oblivious of their potential significance when interpreted as HTML.

The test had been introduced on December 3rd, 2017, 570 commits ago. Every commit on master and in PRs since introduction in that commit triggers these tests to run for every supported PHP version. Additional test cycles get run on developers local environments during working on commits. Hence the test suite must have run a few thousand times, testing 100 random strings each time. And only after more then 2 years a string containing `<a` got generated, triggered the error condition and 22 shrinks later the smallest failing test case was presented as:

```
Failed after 65 tests and 22 shrinks. rngState: 8b8f0d4ec2a67139b5; Counterexample: "image/png"; ""; "\u0000"; "<a"; "";
```

Discussion about a potential problem with that code, did spark last September, when the vulnerable code part was changed to the one before before the current fix, but was incorrectly judged as not being a problem, because all of our translation strings are hardcoded. The fact that we do actually add some untrusted user-provided content, wasn't considered at that point.

It should also be mentioned, that the coverage report released for version 1.3.1 did highlight the line that caused this vulnerability as not being covered during testing:

Coverage report for version 1.3.1 showing missing test coverage for line causing vulnerability

So, in conclusion, it is great to have all of these tools at our disposal, but the code quality would benefit a lot more from having more eyeballs and brains on it.

Timeline

- 2019-12-24 – Property based unit test fails in a commit pushed to a PR.
- 2019-12-25 – Issue investigated, preliminary patch and issue description published.
- 2019-12-30 – Further investigations, proof-of-concept exploit demonstrated on a vulnerable test instance.
- 2020-01-03 – Discussed broader mitigation of user provided content injections, reviewed other possible cases.
- 2020-01-04 – Published a second patch based on review, escapes HTML in translation.
- 2020-01-05 – Started writing vulnerability report.
- 2020-01-07 – Backported fix for 1.2.1.
- 2020-01-11 – Release published.
- 2020-01-11 – Vulnerability details published.

Severity

Moderate

CVE ID

CVE-2020-5223

Weaknesses

No CWEs