79

Potential unprivileged Stored XSS through wp_targeted_link_rel

Share:

State	Resolved (Closed)
Disclosed	**January 8, 2020 9:42pm +0530**
Reported To	**WordPress**
Asset	WordPress Core (Source code)
CVE ID	CVE-2019-16773
Weakness	Cross-site Scripting (XSS) - Stored
Bounty	$650
Severity	High (7 ~ 8.9)
Participants	
Visibility	Disclosed (Full)

Collapse

TIMELINE · EXPORT

simonscannell submitted a report to **WordPress**. Mar 15th (about 1 year ago)

The user description is vulnerable to a Stored XSS via an attribute injection. At fault is the `wp_targeted_link_rel()` filter that parses attributes regardless of their position.

```
function wp_targeted_link_rel( $text ) {
    // Don't run (more expensive) regex if no links with targets.
    if ( stripos( $text, 'target' ) !== false && stripos( $text, '<a ' ) !== false ) {
        $text = preg_replace_callback( '|<a\s([^>]*target\s*=[^>]*)>|i', 'wp_targeted_link_rel_callback',
    }
```

It essentially just parses the attribute string of all `<a>` tags and passes them to the preg replace callback.

```
function wp_targeted_link_rel_callback( $matches ) {
    $link_html = $matches[1];
    $rel_match = array();
...
// Value with delimiters, spaces around are optional.
    $attr_regex = '|rel\s*=\s*?(\\\{0,1}["\'])(.*?)\\1|i';
    preg_match( $attr_regex, $link_html, $rel_match );

    if ( empty( $rel_match[0] ) ) {
        // No delimiters, try with a single value and spaces, because `rel =  va"lue` is totally fine...
        $attr_regex = '|rel\s*=(\s*)([^\s]*)|i';
        preg_match( $attr_regex, $link_html, $rel_match );
    }
```

As can be seen it then uses a regex to parse the `rel` attribute, its value and its delimeter from the string.

If the rel attribute is found, the following happens:

```
    if ( ! empty( $rel_match[0] ) ) {
        $parts     = preg_split( '|\s+|', strtolower( $rel_match[2] ) );
        $parts     = array_map( 'esc_attr', $parts );
        $needed    = explode( ' ', $rel );
        $parts     = array_unique( array_merge( $parts, $needed ) );
        $delimiter = trim( $rel_match[1] ) ? $rel_match[1] : '"';
        $rel       = 'rel=' . $delimiter . trim( implode( ' ', $parts ) ) . $delimiter;
        $link_html = str_replace( $rel_match[0], $rel, $link_html );
```

As you can see the value of the `rel` attribute is splitted by whitespaces and each part is then escaped. The targeted `rel` value is then added to the alread existing ones and put back together.

Most importantly, are the following line:

```
        $delimiter = trim( $rel_match[1] ) ? $rel_match[1] : '"';
        $rel       = 'rel=' . $delimiter . trim( implode( ' ', $parts ) ) . $delimiter;
        $link_html = str_replace( $rel_match[0], $rel, $link_html );
```

if the delimeter is empty (e.g. when `rel=abc` has no quotes), the delimer becomes `"`. The original rel attribute is then replaced with the new one.

This is a problem since the following payload:

`<a title=" target='xyz' rel=abc ">PoC</a>`

would turn into

`<a title=" target='xyz' rel="abc" ">PoC</a>` Note that an additional `"` has been injected and the title attribute has been escaped.

This is because the regex to match the rel attribute ignores the position of the `rel` attribute within the attribute string. The above payload shows how the rel attribute is placed within a double quoted attribute. Since no delimeter is set, the delimer becomes a double quote and when the rel attribute is inserted back into the string, the double quote is injected.

I recommend using something like `parse_shortcode_atts()` as in `wp_rel_nofollow()` to prevent this from happening.

By abusing the attribute injection, it is easily possible to create a Stored XSS payload.

Tge `wp_targeted_link_rel()` filter is not only called on the user description, however, this is where it becomes exploitable. This is because this vulnerable filter is added before the `kses` filters are added, which means that the injected attribute would be caught by `wp_post_kses()`. The user description is the only exception where the kses filters are called before `wp_targeted_link_rel()` is called.

`<a href="#" title=" target='abc' rel= onmouseover=alert(/XSS/) ">This is a PoC for a Stored XSS</a>`

Proof of Concept

The following will demonstrate how a normal forum user can achieve stored XSS on their profile page in BuddyPress
[redacted]

1. This works if the Bio of forum users is displayed in their profile page. Log in as an administrator and go to Appearence -> Customize and then BuddyPress Nouveu -> Member front page and make sure that displaying the user bio is enabled
2. Create a normal forum user account
3. Login and edit your profile. Paste `<a href="#" title=" target='abc' rel= onmouseover=alert(/XSS/) ">This is a PoC for a Stored XSS</a>` as your user description
4. visit your profil and hover over the link.

Impact

The Impact of this can vary from site to site. I have shown how this can be exploited in BuddyPress as a mere, normal forum user. Since you can also inject a style attribute and make the link span over the entire page, one can turn this into a wormable Stored XSS in BuddyPress.

Basically every plugin or forum is affected that displays the user description.

vortfu changed the status to ○ **Triaged**. Mar 22nd (about 1 year ago)
Thanks for the report, we're currently testing a patch for the issue and will keep you updated with any developments.

simonscannell posted a comment. Updated Jan 8th (5 months ago)
[redacted]

simonscannell posted a comment. Nov 19th (6 months ago)
Hi @vortfu ,

Are there any updates on the patch?

whyisjake posted a comment. Nov 20th (6 months ago)
Hey @simonscannell,

We put together this patch that aims to fix this problem. Could you please review?

Cheers,

@whyisjake

simonscannell posted a comment. Nov 25th (6 months ago)
Hi @whyisjake ,

Seems effective to me!

Best,
Simon

whyisjake posted a comment. Dec 11th (6 months ago)
Hey there @simonscannell,

Do you have a GitHub account? Would like to add you to the Security Advisory there.

Also, would the "Props to Simon Scannell of RIPS Technologies" still work for the release on Thursday?

Cheers,

@whyisjake

simonscannell posted a comment. Dec 11th (6 months ago)
Hi @whyisjake ,

My github account is https://github.com/scannells thank you!

Yes, please do use this props for all Security issues reported by me!

Best regards,
Simon

ehtis posted a comment. Dec 16th (6 months ago)
Hi Simon,

This went out in https://wordpress.org/news/2019/12/wordpress-5-3-1-security-and-maintenance-release/

Working on getting the bounty released. Thank you for your great research as always! :)

WordPress rewarded simonscannell with a **$600** bounty and a **$50** bonus. Dec 18th (6 months ago)

ehtis closed the report and changed the status to ○ **Resolved**. Dec 18th (5 months ago)

simonscannell posted a comment. Updated Jan 8th (5 months ago)
Thank you very much!

I was told [redacted]
Is also in the queue, but since it was Not included in this round I just wanted to make sure there are no confusions, thank you!

ehtis posted a comment. Dec 20th (5 months ago)
Yep! Also working on getting that released.

simonscannell posted a comment. Dec 20th (5 months ago)
Thank you for taking care of this!

simonscannell requested to disclose this report. Jan 4th (5 months ago)

whyisjake agreed to disclose this report. Jan 8th (5 months ago)
@simonscannell,

I just published this security advisory with associated CVE: CVE-2019-16773

https://github.com/WordPress/wordpress-develop/security/advisories/GHSA-xvg2-m2f4-83m7

Cheers,

@whyisjake

This report has been disclosed. Jan 8th (5 months ago)

simonscannell posted a comment. Jan 8th (5 months ago)

Hi @whyisjake ,

Thank you! I like the direction WordPress is going with full disclosure.

All the best,
Simon

peterwilsoncc updated CVE reference to **CVE-2019-16773**. Jan 25th (4 months ago)