Ram Gall July 21, 2020

High Severity Vulnerability Patched in TC Custom JavaScript

On June 12, 2020, Wordfence Threat Intelligence discovered an unauthenticated stored Cross-Site Scripting(XSS) vulnerability in TC Custom JavaScript, a WordPress plugin with over 10,000 installations.

Wordfence Premium customers received a new firewall rule to provide protection against attacks targeting this vulnerability the same day. Wordfence users still using the free version received this rule after 30 days, on July 12, 2020.

We attempted to contact the plugin's developer the same day, on June 12, 2020, but we did not receive a response. After 10 days without an initial response, we contacted the WordPress Plugins team on June 22, 2020. An initial patch was released the next day, on June 23, 2020, and a full patch was released on June 29, 2020.

Description: Unauthenticated Stored Cross-Site Scripting(XSS)
Affected Plugin: TC Custom JavaScript
Plugin Slug: tc-custom-javascript
Affected Versions: < 1.2.2
CVE ID: CVE-2020-14063
CVSS Score: 8.3(high)
CVSS Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:L/A:L
Fully Patched Version: 1.2.2

TC Custom JavaScript is a WordPress plugin that allows site owners to add custom JavaScript to every page on a site and emphasizes a minimalist approach. While this kind of functionality can be incredibly useful, it can also be incredibly dangerous without proper access controls.

This plugin ran the `TCCJ_Core_Content::update` function immediately upon startup. This was unusual, as most WordPress plugins register functions to be run at predictable points in the WordPress load process.

As a result, the `update` function ran before WordPress access control functions, such as capability checks and nonce verification, could be used. This, combined with the lack of capability checks and nonce verification on the function, made it possible for unauthorized visitors to use the `update` function.

The `update` function used a companion function, `has_update_request`, to check if the `tccj-update` `POST` parameter was supplied and set to `Update`. If this check passed, then the `update` function would accept the contents of the `tccj-content` `POST` parameter and add it to the database.

```
class TCCJ_Core_Content {
    public static function update() {
        if ( self::has_update_request() ) {
            if ( get_magic_quotes_gpc() )
                $tccj_content = stripslashes( $_POST['tccj-content'] );
            else
                $tccj_content = $_POST['tccj-content'];

            // Sanitizing data before insert to database
            $tccj_content = wp_check_invalid_utf8( $tccj_content, true );
            $tccj_content = htmlentities( $tccj_content );

            if ( ! get_magic_quotes_runtime() )
                $tccj_content = addslashes( $tccj_content );

            update_option( 'tccj_content', $tccj_content );
        }
    }

    private static function has_update_request() {
        if ( isset( $_POST['tccj-update'] ) && ( $_POST['tccj-update'] == 'Update' ) )
            return true;
        else
            return false;
    }
}
```

While the `update` function did run `htmlentities` on the content provided by `tccj-content` before storing it, this didn't provide any additional safety, as the `TCCJ_Core_Frontend::print_script_in_footer` function used to display the added script not only used `html_entity_decode` on the stored content but also added `<script>` tags around it.

```
class TCCJ_Core_Frontend {
    public static function print_script_in_footer() {
        //$tccj_content = sanitize_text_field( get_option( 'tccj_content', '' ) );
        $tccj_content = get_option( 'tccj_content', '' );
        $tccj_content = stripslashes( $tccj_content );
        $tccj_content = html_entity_decode( $tccj_content );

        if ( $tccj_content != '' ) {
            echo '<script type="text/javascript">' . $tccj_content . '</script>';
        }
    }
}
```

As such, an attacker could send a `POST` request to any location on a vulnerable site with the `tccj-update` parameter set to `Update` and the `tccj-content` parameter set to malicious JavaScript, and this JavaScript would display in the footer of every page on the site.

Malicious JavaScript of this type can be used to redirect visitors to malvertising sites or steal payment information. Even worse, it can detect when an administrator visits the site and send a request on their behalf to infect files with a backdoor or possibly create a new, malicious administrator user account leading to takeover of the entire site.

Timeline

rule available to Wordfence Premium users and attempt to make contact with the plugin's developer.
June 22, 2020 – After failing to make contact with the plugin's developer, we contact the WordPress Plugins team to notify them of the vulnerability.
June 23, 2020 – The plugin developer releases an initial patch which is still vulnerable to CSRF attacks.
June 29, 2020 – A fully patched version of the plugin is released.
July 12, 2020 – Firewall rule becomes available to free Wordfence users.

Conclusion

In today's article, we reviewed an unauthenticated stored Cross-Site Scripting(XSS) vulnerability in the TC Custom JavaScript plugin. This flaw has been fully patched in version 1.2.2 and we strongly recommend updating to this version as soon as possible. Sites running Wordfence Premium have been protected against these vulnerabilities since June 12, 2020, while sites still using the free version of Wordfence received the firewall rule on July 12, 2020.

Special thanks to Wordfence QA Lead Matt Rusnak, who initially discovered the vulnerability.

Comments

No Comments