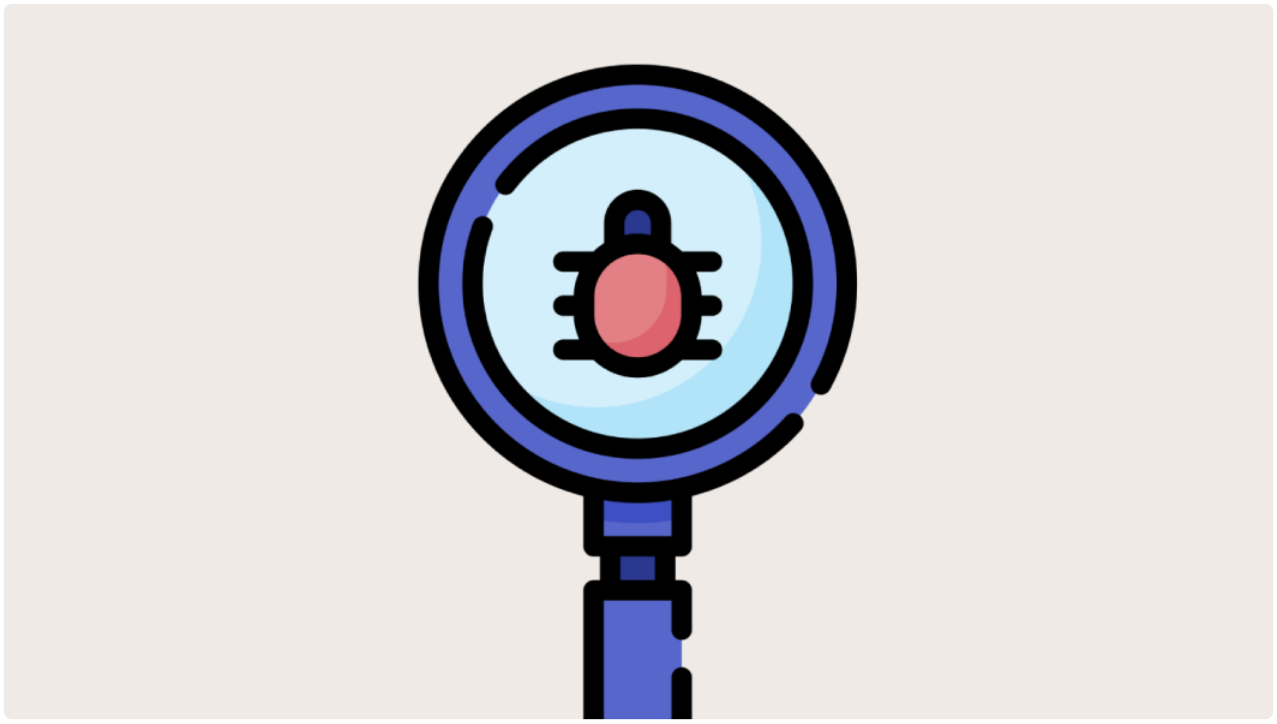
CVE CYBERSECURITY

Technical write-up on CVE-2021-24313

BASTIJN OUWENDIJK – 31 MAY 2021

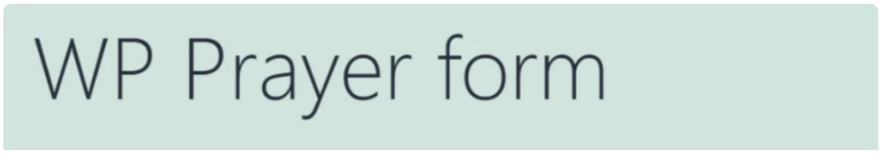


While surfing through the sea of WordPress plugins, I stumbled upon a divine plugin. A plugin where users can request prayers, providing users the opportunity to ask for salvation. However, this plugin carried a sin of its own. In the following write-up, I'll explain a vulnerability that I found in the WP Prayer WordPress plugin CVE-2021-24313. This vulnerability allows an authenticated attacker to store malicious JavaScript code into the 'requested prayers' list where requester prayers of users are shown.

WP Prayer is a WordPress plugin where authenticated users can submit prayer requests through a prayer form. The plugin can list all the requested prayers by users on a post or page. **WP Prayer version 1.6.1 and earlier** contains an **Authenticated Stored Cross-Site Scripting (XSS) vulnerability**.

How the vulnerability works

Let's dive into the details of how this authenticated stored XSS works! To submit a prayer request a user has to be authenticated. I created a WordPress user account on the website with the role Subscriber, which is the role with the least privileges within WordPress. When logged into the website with the subscriber role, the prayer request form looks like this.


WP Prayer form

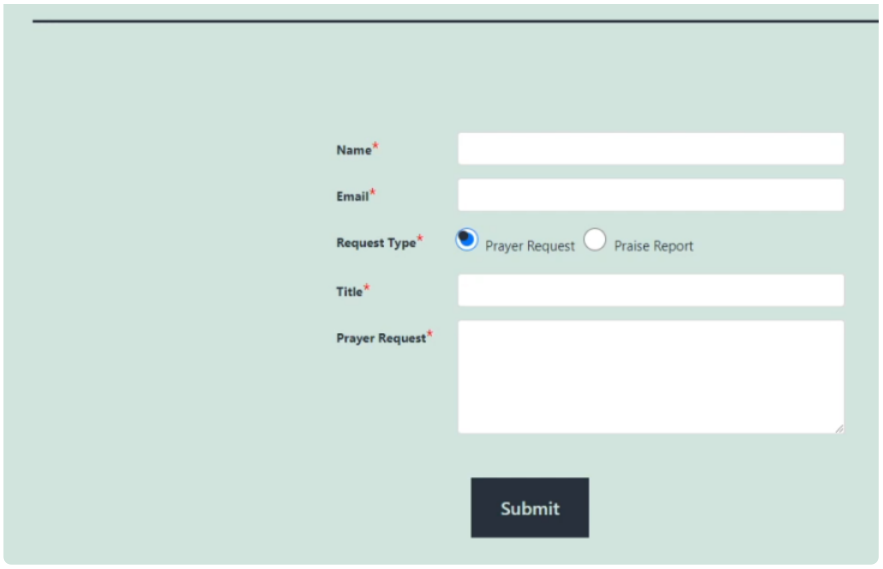


Figure 01. The prayer request form

The form field where a prayer request can be made contains the following PHP code:

```
$form->add_element( 'textarea', 'prayer_messages', array(
 'label' => ( 'Prayer Request', WPE_TEXT_DOMAIN ),
 'value' => (isset( $data['prayer_messages'] ) and ! empty( $data['prayer_messages'] )) ?
$data['prayer_messages']  : '', 'desc' => ( 'Enter here your message.', WPE_TEXT_DOMAIN
),
 'textarea_rows' => 10,
 'required' => true,
 'textarea_name' => 'prayer_messages',
 'class' => 'form-control',
 ));
```

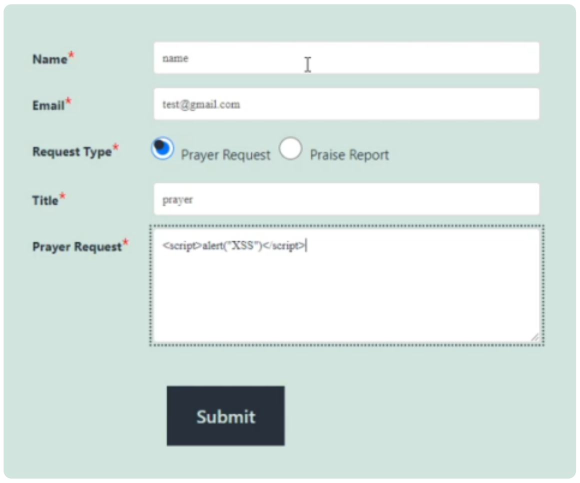
The code in line 3 checks whether data is present in the text area of the 'Prayer Request' field. If the text area field is not empty, it sets the string in the text area to the variable `$data['prayer_messages']` and when the form is submitted, does a POST request to the website.

Next, the received POST data from the prayer request forms flows to the data model, which has a function to save the user's prayer request to the database. If the received `prayer_messages` is declared and different from null, the slashes of the string are removed, as can be seen in lines 22-23 in the code below. When completed, the prayer request is successfully stored in the database.

What's interesting is that the `nonce`, `prayer_title` and `request_type` variables are sanitized by the function `sanitize_text_field`, while the `prayer_messages` variable is not.

```
public function save()
{
    $entityID = '';
    if (isset($_REQUEST['_wpnonce'])) {
        $nonce = sanitize_text_field(wp_unslash($_REQUEST['_wpnonce']));
    }
    if (isset($nonce) and ! wp_verify_nonce($nonce, 'wpgmp-nonce')) {
        die('Cheating...');
    }

    if (isset($_POST['request_type']) && $_POST['request_type'] == 'prayer_request') {
        $this->validations['prayer_title'] = array('req' => __('Please enter title.',WPE_
TEXT_DOMAIN));
    }
    $this->validations['prayer_messages'] = array('req' => __('Prayer Request',WPE_TEXT_D
OMAIN));
    $this->verify($_POST);
    if (is_array($this->errors) and ! empty($this->errors)) {
        $this->throw_errors();
    }
    if (isset($_POST['entityID'])) {
        $entityID = intval(wp_unslash($_POST['entityID']));
    }
    if (isset($_POST['prayer_messages'])) {
        $data['prayer_messages'] = wp_unslash($_POST['prayer_messages']);
    }
    $data['prayer_title'] = sanitize_text_field(wp_unslash($_POST['prayer_title']));
    $data['prayer_author'] = get_current_user_id();

    $data['prayer_status'] = 'approved';
    $data['prayer_time'] = date('Y-m-d H:i:s');
    $lxt_options = get_option('_wpe_prayer_engine_settings');
    $lxt_options = unserialize($lxt_options);

    if ( ! empty($lxt_options) && array_key_exists('wpe_disapprove_prayer_default', $lxt_
options)) {

        $data['prayer_status'] = (filter_var($lxt_options['wpe_disapprove_prayer_defaul
t'],
            FILTER_VALIDATE_BOOLEAN)) ? 'pending' : 'approved';
    }


    $data['request_type'] = sanitize_text_field(wp_unslash($_POST['request_type']));

    if ($entityID > 0) {
        $where[$this->unique] = $entityID;
    } else {
        $where = '';
    }
    $result = FlipperCode_Database::insert_or_update($this->table, $data, $where);
    if (false === $result) {
        $response['error'] = __('Something went wrong. Please try again.', WPE_TEXT_DOMAI
N);
    } elseif ($entityID > 0) {
        $response['success'] = __('Prayer updated successfully', WPE_TEXT_DOMAIN);
    } else {
        $response['success'] = __('Prayer added successfully.', WPE_TEXT_DOMAIN);
    }

    return $response;
}
```

The string stored in the ‘prayer_messages’ variable is not sanitized or validated for special characters besides slashes; a security sin. This makes it possible to store JavaScript payloads in the database. Let’s submit the following prayer request.



Name*	name
Email*	test@gmail.com
Request Type*	[✓ Prayer Request] Praise Report
Title*	prayer
Prayer Request*	<script>alert("XSS")</script>

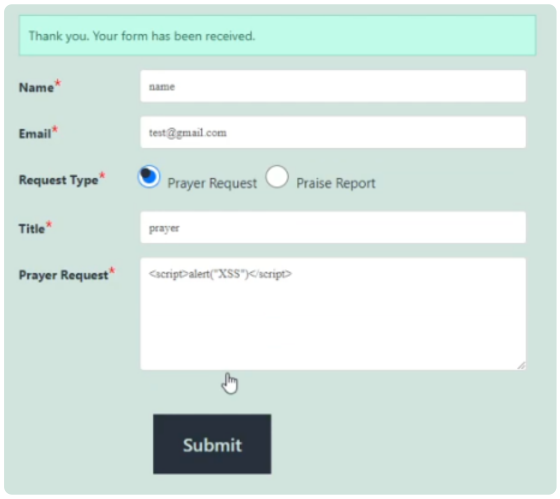
Figure 02. The prayer request form – Step 1: Submit the form including a JavaScript payloadm

This is the JavaScript payload:

```
<script>alert("XSS")</script>
```

This payload, when executed, will create an alert in the browser containing the text ‘XSS’. However, between the script tags, you can put other JavaScript code.

Let’s submit the prayer request including our payload, we get a message saying the form was received. Our prayer request is now saved in the database.


Thank you. Your form has been received.

Name*	name
Email*	test@gmail.com
Request Type*	[✓ Prayer Request] Praise Report
Title*	prayer
Prayer Request*	<script>alert("XSS")</script>

Figure 03. The prayer request form – Step 2: We successfully submitted the form including our payload

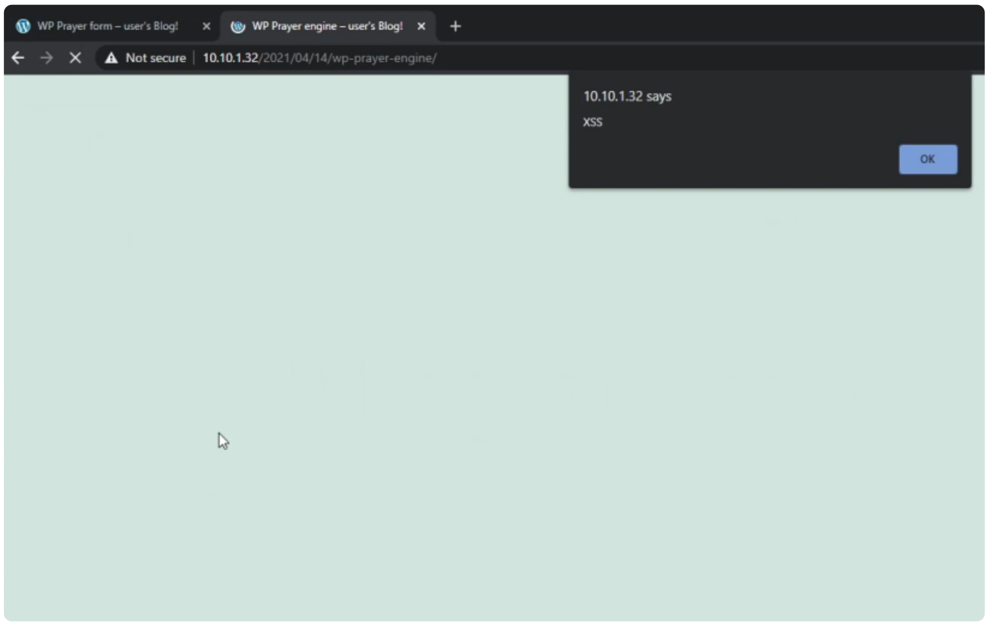
Let’s look at how the stored prayer requests are loaded into the webpage. In the shortcode of the WP Prayer Engine, the following PHP code loads the stored prayer request information from the database into the HTML of the webpage. However, there is no validation of the data that is being loaded into the webpage.

```
echo ' <h5>'.$pray->prayer_title.'</h5>'.nl2br($pray->prayer_messages).'<div class="postm
eta">';
```

Let’s see how the HTML of the webpage looks when the PHP code is executed and our prayer request is loaded from the database into the HTML:

```
<h5>prayer</h5><script>alert("XSS")</script><div class="postmeta">
```

This means we successfully inserted a JavaScript payload into the webpage. When the webpage loads it will execute the JavaScript code and an alert with the text ‘XSS’ is shown in the browser.


WP Prayer form – user's Blog!
WP Prayer engine – user's Blog!
Not secure
10.10.1.32/2021/04/14/wp-prayer-engine/

10.10.1.32 says

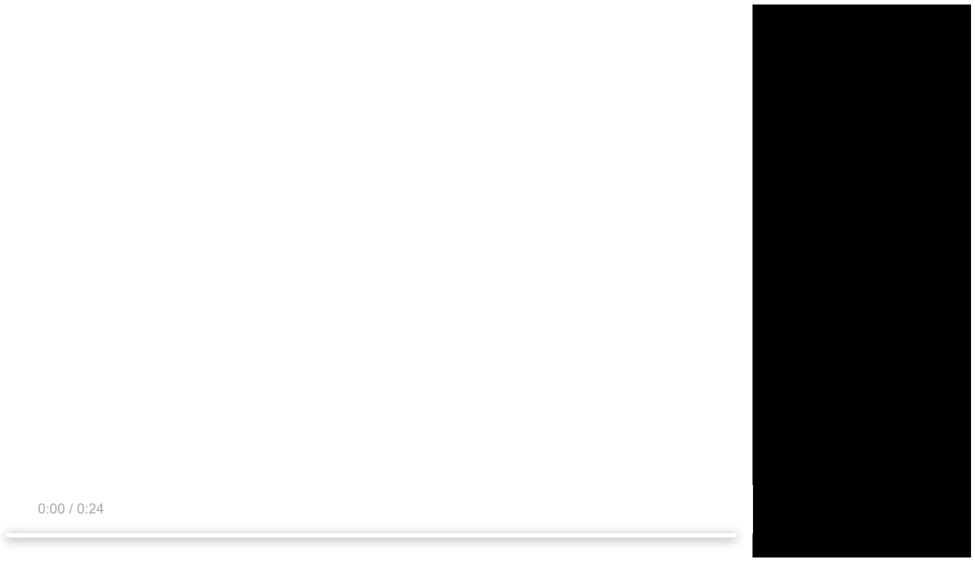
XSS

Figure 04. The prayer request engine – Step 3: Execute the payload by loading the prayer request engine which lists all the stored requested prayers

Impact

The authenticated stored XSS in WP Prayer 1.6.1 and earlier poses a risk for visitors of the webpage where prayer requests are listed. For example, stored XSS can be used by a malicious actor to potentially steal cookies or sensitive data from visitors.

Proof of Concept



Video 01. Proof of Concept video of authenticated stored XSS in WP Prayer version 1.5.5

Conclusion

WP Prayer 1.6.1 and earlier versions contain an authenticated stored XSS vulnerability. This makes it possible to store malicious JavaScript code on web pages where prayer requests are listed.

The vulnerability exists because the prayer message field of the prayer request form does not sanitize and validate user input, therefore code can be saved to the database. In addition, the prayer message field string that is saved in the database is loaded into the webpage without special characters (such as <, >, &, “, and ‘) being converted to HTML entities, which leads to the code being loaded into the webpage. Therefore, the code will be loaded and executed into the webpage when prayer requests are listed.

With this responsible disclosure, I hope I provided salvation by saving this plugin from a security sin.

Timeline

15 April, 2021: WPScan – Vendor contacted
21 April, 2021: WPScan – Escalated to WordPress
14 May, 2021: Vulnerability fixed by vendor
17 May, 2021: Disclosure on WPScan and CVE-2021-24313 assigned
31 May, 2021: Proof of Concept disclosure

References

https://wpscan.com/vulnerability/c7ab736d-27c4-4ec5-9681-a3f0dda86586
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24313
https://wordpress.org/plugins/wp-prayer/
https://www.exploit-db.com/exploits/49921

You may also like

CVE CYBERSECURITY

Technical write-up on CVE-2022-2753

BASTIJN OUWENDIJK – 30 SEP 2022

Write-up about finding an Unauthenticated Stored Cross-Site Scripting (XSS) vulnerability in Ketchup Restaurant Reservations version 1.0.0 and earlier.

OSWA CERTIFICATE CYBERSECURITY

BASTIJN OUWENDIJK – 19 JUL 2022

Write-up about my OSWA preparations, my experience with taking the exam, and concluding thoughts about this certificate.

EWPT CERTIFICATE CYBERSECURITY

My journey to becoming an eWPT

BASTIJN OUWENDIJK – 11 AUG 2021

Write-up about my eWPT preparations, my experience with taking the exam, and concluding thoughts about this certificate.

Bastijn Ouwendijk

I work in cyber security and like to break code.