Ram Gall

August 3, 2020

Newsletter Plugin Vulnerabilities Affect Over 300,000 Sites

On July 13, 2020, our Threat Intelligence team was alerted to a recently patched vulnerability in Newsletter, a WordPress plugin with over 300,000 installations. While investigating this vulnerability, we discovered two additional, more serious vulnerabilities, including a reflected Cross-Site Scripting(XSS) vulnerability and a PHP Object Injection vulnerability.

We reached out to the plugin's author on July 15, 2020, and received a response the next day. After fully disclosing the vulnerability on July 16, 2020, the plugin's author released a patch the next day, on July 17, 2020.

A firewall rule to protect against the Reflected Cross-Site Scripting vulnerability was released to Wordfence Premium customers on July 15, 2020 and will become available to free Wordfence users 30 days later, on August 14, 2020.

Although the PHP Object Injection vulnerability would require additional vulnerable software to be installed, and our built-in PHP Object Injection protection would have protected against the most common exploits, we determined that a bypass was possible. Out of an abundance of caution, we created an additional firewall rule and released it to Wordfence Premium users on July 28, 2020. The PHP Object Injection firewall rule will become available to free Wordfence users on the same date as the XSS rule for this plugin, on August 14, 2020.

Description: Authenticated Reflected Cross-Site Scripting(XSS)
Affected Plugin: Newsletter
Plugin Slug: newsletter
Affected Versions: < 6.8.2
CVE ID: CVE-2020-35933
CVSS Score: 6.5(Medium)
CVSS Vector: CVSS:3.0/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:L
Fully Patched Version: 6.8.2

The Newsletter plugin includes a full-featured visual editor that can be used to create visually appealing newsletters and email campaigns. It uses an AJAX function, `tnpc_render_callback`, to display edited blocks based on a set of options sent in the AJAX request. Unfortunately, the vulnerable versions did not filter these options, but passed them onto a second function, `restore_options_from_request` which used multiple methods to decode options that were passed in before displaying them using the `render_block` function.

```
function tnpc_render_callback() {
    $block_id = $_POST['b'];
    $wrapper = isset($_POST['full']);
    $options = $this->restore_options_from_request();

    $this->render_block($block_id, $wrapper, $options);
    wp_die()
```

As such, it was possible for an attacker to get malicious JavaScript to display in multiple ways. The simplest method would involve sending a `POST` request to `wp-admin/admin-ajax.php` with the `action` parameter set to `tnpc_render`, the `b` parameter set to `html`, and the `options` parameter set to arbitrary JavaScript. Alternatively, a similar request with the `options` parameter set to an empty array `options[]=` and the `encoded_options` parameter set to a base64-encoded JSON string containing arbitrary JavaScript would also result in JavaScript being rendered in a logged-in user's browser.

```
if (isset($_POST['encoded_options'])) {
    $decoded_options = $this->options_decode($_POST['encoded_options']);
```

```
function options_decode($options) {

    // Start compatibility
    if (is_string($options) && strpos($options, 'options[') !== false) {
        $opts = array();
        parse_str($options, $opts);
        $options = $opts['options'];
    }
    // End compatibility

    if (is_array($options)) {
        return $options;
    }

    $tmp = json_decode($options, true);
    if (is_null($tmp)) {
        return json_decode(base64_decode($options), true);
    } else {
        return $tmp;
    }
}
```

We discussed Reflected XSS vulnerabilities in a previous post. Despite the fact that they require an attacker to trick a victim into performing a specific action (such as clicking a specially crafted link), they can still be used to inject backdoors or add malicious administrative users. If an attacker tricked a victim into sending a request containing a malicious JavaScript using either of these methods, the malicious JavaScript would be decoded and executed in the victim's browser.

Description: PHP Object Injection
Affected Plugin: Newsletter
Plugin Slug: newsletter
Affected Versions: < 6.8.2
CVE ID: CVE-2020-35932
CVSS Score: 7.5(High)
CVSS Vector: CVSS:3.0/AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:H
Fully Patched Version: 6.8.2

Although the Newsletter editor did not allow lower-level users to save changes to a given newsletter, the same [illegible] AJAX function was still accessible to all logged-in users, including subscribers. This introduced passed in via the `options[inline_edits]` parameter. As such, an attacker logged-in as a subscriber could send a `POST` request to `wp-admin/admin-ajax.php` with the `action` parameter set to `tpnc_render` and the `options[inline_edits]` parameter set to a serialized PHP object.

```
if (isset($_POST['options']) && is_array($_POST['options'])) {
    // Get all block options
    $options = stripslashes_deep($_POST['options']);

    // Deserialize inline edits when
    // render is preformed on saving block options
    if (isset($options['inline_edits']) && is_serialized($options['inline_edits'])) {
        $options['inline_edits'] = unserialize($options['inline_edits']);
    }
```

Although the Newsletter plugin itself did not use any code that would allow additional exploitation, this vulnerability could be used to inject a PHP object that might be processed by code from another plugin or theme and used to execute arbitrary code, upload files, or any number of other tactics that could lead to site takeover.

How does PHP Object Injection Work?

PHP can make use of a method called "serialization" to store complex data. In most cases serialized data consists of key => value arrays, for example:

`a:2:{s:11:"productName";s:5:"apple";s:7:"price";i:10;}`

This serialized data sample includes a `productName` property which is set to `apple` and a `price` property which is set to `10`.

Serialized data is useful for storing settings in bulk, and many WordPress settings are stored as serialized data. Unfortunately, serialized data can also cause a security issue because it can be used to store PHP objects.

What are PHP objects?

Most modern PHP code is object oriented, meaning that code is organized into "classes." These classes act like a basic template containing both variables (referred to as "properties") and functions (referred to as "methods"). A running program can then create "objects" based on these templates, or classes. This creates not only a clear, concise structure for handling data, making code easier to maintain, it allows the same code to be reused for multiple similar tasks.

For instance, an online store could use a single class for products with properties(variables) including `$price` and `$productName`, and create a different object for each product. Each object would use the same function(method) to calculate tax, but could use a different price and product name.

If a plugin unserializes data provided by users without sanitizing that user's input, then an attacker can send a specially crafted payload that would be unserialized into a PHP object.

On its own, an injected PHP object is not particularly dangerous. This changes, however, if the class it is based on uses so-called "magic methods".

What are magic methods?

Magic methods are special functions that can be added to a class that describe what it should do when certain events happen.

For instance, the `__destruct` function is used in many classes to "clean up" once an object is done being used, and in many cases it does this by deleting files.

Here's a very basic example of a vulnerable class that calculates product prices, stores a log, and deletes the log when it's done:

```
class Product{

    public $price;
    public $productName;
    public $savedPriceFile;

    function __construct($price, $productName){
        $this->price=$price;
        $this->productName=$productName;
        $this->savedPriceFile=$productName."pricefile.log";
    }

    function calculateTotal($quantity) {
        $total=$this->price * $quantity;
        echo ($total);
        file_put_contents($this->savedPriceFile, $total);
    }

    function __destruct(){
        unlink($this->savedPriceFile);
    }

}
```

If this code was running on a site that also had a PHP Object Injection vulnerability, an attacker could delete the `wp-config.php` file containing the WordPress site's core configuration settings by sending a payload similar to the following:

`O:7:"Product":3:{s:5:"price";i:2;s:11:"productName";s:6:"apples";s:14:"savedPriceFile";s:13:"wp-config.php";}`

This would inject a `Product` object, with the `$productName` set to `apples`, the `$price` set to `2`, and a `$savedPriceFile` property set to `wp-config.php`. Even though the object might not be used by anything else, eventually the `__destruct` function would run, deleting whatever `$savedPriceFile` was set to. In this case, the deletion of the `wp-config.php` file would reset the site and allow an attacker to take over by pointing the site's new configuration to a remote database under their control.

Successfully exploiting this chain of events, also known as a "POP chain", does require some degree of effort, since it requires:

- Code that unserializes user input (an Object Injection vulnerability).
- Code that uses a magic method in an insecure way.
- Both of these need to be loaded at the same time.

Due to the fact that many plugins and themes load some, or all, of their classes on each request to the site, this is not as great of a restriction as it might appear. Additionally, although insecure usage of these "magic methods" is less common than it was in the past, such usage is not considered a vulnerability on its own since it requires the presence of a PHP Object Injection vulnerability to exploit.

Finally, although an attacker might need to know which plugins are installed in order to tailor their attack to a given POP chain, it is often fairly simple to determine this with scanning tools. The good news is that such vulnerabilities are difficult to automatically exploit in bulk, except in cases where a PHP Object Injection vulnerability and an insecure magic method are both used in the same plugin.

Timeline

July 13, 2020 – Our Threat Intelligence Team begins investigating a recently patched vulnerability in the Newsletter plugin.
July 14, 2020 – During our investigation, we discover 2 unpatched vulnerabilities.
July 15, 2020 – We release a firewall rule for the reflected XSS vulnerability to Wordfence Premium users and reach out to the plugin's author.

July 16, 2020 – We receive a response from the plugin's author and provide full disclosure.
July 17, 2020 – Plugin author releases a patch for both vulnerabilities.
August 14, 2020 – Both firewall rules become available to free Wordfence users.

Conclusion

In this blog post, we discussed 2 vulnerabilities in the Newsletter plugin, including a reflected XSS vulnerability and a PHP Object Injection vulnerability. We also explained what PHP Object Injection vulnerabilities are and how they can be exploited.

We strongly recommend updating to the latest version of the Newsletter plugin as soon as possible. As of this writing, that is version 6.8.3.

Wordfence Premium users have been protected against the majority of potential attacks since July 15, 2020, and have been fully protected since July 28, 2020. Sites still running the free version of Wordfence will receive firewall rules protecting against both vulnerabilities on August 14, 2020.

Special thanks to Stefano Lissa & The Newsletter Team for their rapid response in patching these vulnerabilities.

Comments

6 Comments

Sisir *
August 3, 2020
11:45 am

Pardon me the ignorance.

If the function hooked into the hook "wp_ajax_" hook not in "wp_ajax_nopriv_" hook, does it mean the code is equally vulnerable (considering a site which doesn't accept user registration)? because in that case only logged in users can send the attack.

Ram Gall *
August 3, 2020
12:09 pm

Hi Sisir

You're correct that both attacks require a logged-in user. However, the Reflected XSS vulnerability requires socially engineering a victim into clicking a link, so the victim would be logged in, rather than the attacker. Neither vulnerability was conducive to bulk exploitation, but both could result in serious consequences if a targeted attack was used.

zean *
August 4, 2020
12:03 am

This is a problem for only the "Newsletter " plugin right?

Ram Gall *
August 4, 2020
3:15 pm

Hi zean,

That's correct. It has also been patched so if you're using the latest version of the Newsletter plugin you're safe.

Ama *
August 5, 2020
7:52 am

Great article - thanks. If I update the plugin will I lose settings in the leads / follow up addons, including where subscribers are in a series of emails?

Ram Gall *
August 5, 2020
7:59 am

Hi Ama,

You should not lose any settings by updating - all of that information is stored in your site's database so it is preserved when your site is updated.

you@example.com

SIGN UP