mcdruid / **lightning_opencart_object_injection.md** Secret

Last active 2 hours ago

Code Revisions 2

Lightning Opencart module PHP Object Injection

`lightning_opencart_object_injection.md`

Summary

The OpenCart Lightning module has a PHP Object Injection vulnerability as a result of Deserialization of Untrusted Data.

(POP/) Gadget Chains exist in OpenCart (3 and 4) which allow Object Injection vulnerabilities to be exploited, for example to write arbitrary files or achieve Remote Code Execution.

Such an attack could result in the compromise of a site.

Timeline

- 2025-01-26: mcdruid attempts to contact Lightning maintainer

Details of the Module

- https://www.opencart.com/index.php?route=marketplace/extension/info&extension_id=8068
- https://lightning.devs.mx/
- Version details:

```
        <id>OpenCart Lightning</id>
        <version>4.43</version>
        <author><![CDATA[MaxD http://devs.mx]]></author>
```

- Tested with OpenCart 3.0.4.0

Vulnerability Classification

- CWE-502: Deserialization of Untrusted Data
- CAPEC-586: Object Injection
- CVSS (v3): CVSS:3.0/AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H 8.5 High
- CVE: CVE-2025-0974

Vulnerable Code

The module's code is quite heavily obfuscated, and it includes a mechanism to "phone home" and shut itself down if it detects that it is running on "localhost". This makes debugging tricky.

The code is (deliberately) not very human-readable. In order to aid analysis, it can be run through a pretty-printer like https://github.com/lkrms/pretty-php . This doesn't make it easy to read, but it's an improvement.

Global variables, and code in global scope, are used extensively which means it is not always easy to trace the flow.

Piecing together some snippets from two files - `alpha.php` and `zero.php`:

```
$Mk = 'http' . (($_SERVER['SERVER_PORT'] == 443) ? 's://' : '://') .
$_SERVER['HTTP_HOST'] . Wl($_SERVER['REQUEST_URI']);
```

```
        try {
            if (!empty($data))
                $Mrk .= '=' . serialize($data);
        } catch (Exception $Mey) {
        }
        global $Meo;
        $Meo[] = $Mrk;
```

```
    $Mbe = $Mk;
    $Mbe = str_replace('<', '%3C', $Mbe);
    if (strpos($Mbe, '?') or strpos($Mbe, '&'))
        $Mbe .= '&';
    else
        $Mbe .= '?';
```

```
    $Mbe .= 'li_op=gen';
    global $Meo;
    if ($Meo) {
        $Meo = bin2hex(gzcompress(implode('~|~', $Meo), 9));
        $Maf = strrpos($output, "<div id='liaj");
        $output = substr($output, 0, $Maf) .
"\n<script>if(document.cookie.indexOf('li_nr')>=0)document.write('<s'+" .
json_encode('cript src="' . $Mbe . 's&js=1&md=' . $Meo . '&cd=') .
"+Date.now()+'\"></scrip'+'t>')\n</script>\n" . substr($output, $Maf);
    }
```

This code constructs a URL to reference in a `<script>` tag which includes a compressed and encoded value in the `md` parameter - something like:

```
?li_op=gens&js=1&md=[encoded_string]&cd=[timestamp]
```

We can see that the global `$Meo` is an array consisting of one or more values that are key/value pairs in the style of URL parameters where the value is a serialized string. This array is imploded into a string using `~|~` as a separator. That string is then compressed, and encoded with `bin2hex`. The value is then placed in the `md` GET parameter in the assembled URL.

Putting together a few other snippets of code, it looks like when a URL with those components is parsed, the `md` parameter has the reverse processing done on it to turn it back into an array:

```
alpha.php:            $Mz = $_GET['li_op'];
zero.php:             $Mab = $_GET['md'];
```

```
global $Mz, $light_ob, $Mab;

..snip...

    if ($Mz == 'gens' && $Mab) {
        global $Mf, $registry, $Mab;
        $Mab = gzuncompress(hex2bin($Mab));
        $Mcr = explode('~|~', $Mab);

...snip...

            foreach ($Mcr as $Mcb => $Mcs) {
                $Mmo = array();
```

```
                if ($Maf = strpos($Mcs, '=')) {
                    $Mmo = unserialize(substr($Mcs, $Maf + 1));
```

The `$_GET['md']` parameter is decoded (`hex2bin($Mab)`) and uncompressed and then exploded from a string back into an array using the same separator string, `~|~`.

The array is iterated over in a `foreach` loop where each element is checked for the presence of an `=` (which cannot be at index 0) and if it does contain one, everything that follows the `=` is passed to `unserialize()`.

Thus, unsafe input is taken from a GET parameter and passed to PHP's `unserialize()`, which could result in PHP Object Injection.

As a Proof of Concept, we can prepare a simple payload for that GET parameter:

```
php > $o = new DateTime('2025-01-01');
php > $s = serialize($o);
php > $Meo = ['x=' . $s];
php > $Meo = bin2hex(gzcompress(implode('~|~', $Meo), 9));
php > print $Meo;
78daabb0f5b7b2b05272492c490dc9cc4d55b232b6aa2eb632b1524a018a2859175b199959291
```

.. and pass it through the same code the module does:

```
php > $Mab =
'78daabb0f5b7b2b05272492c490dc9cc4d55b232b6aa2eb632b1524a018a2859175b19995929

php > $Mab = gzuncompress(hex2bin($Mab));
php > $Mcr = explode('~|~', $Mab);

php >             foreach ($Mcr as $Mcb => $Mcs) {
php {                 $Mmo = array();
php {                 if ($Maf = strpos($Mcs, '=')) {
php {                     $Mmo = unserialize(substr($Mcs, $Maf + 1));
php {                 }}
```

We should have the Object that was the simulated payload:

```
php > print_r($Mmo);
DateTime Object
(
    [date] => 2025-01-01 00:00:00.000000
    [timezone_type] => 3
```

```
    [timezone] => UTC
)
```

A malicious POP / Gadget Chain payload could be prepared in the same way and passed to this code in the GET parameter.

Mitigation

The call to `unserialize` could be made safer by including the `allowed_classes` option to disable Object Injection:

https://www.php.net/manual/en/function.unserialize.php

For example:

```
$Mmo = unserialize(substr($Mcs, $Maf + 1), ['allowed_classes' => FALSE]);
```

A better mitigation would be to replace the use of serialization with `json_encode` and `json_decode`.

MaxD2 commented 2 hours ago

This vulnerability is fixed since Lightning 4.45 released a year ago.
Please review at last!