Squiz Matrix CMS - Multiple Issues

Security Advisory

Date 9/12/2019

Version: 1.0

Table of Contents

1. Document Control 2

1.1. Document Information 2

1.2. Revision Control 2

2. Background 2

2.1. Introduction 2

2.2. Affected versions 3

2.3. Disclosure Timeline 3

3. Technical Findings 4

3.1. PHP unserialization of user input may result in remote code execution 4

3.2. Arbitrary file deletion and information disclosure in file upload form 6

4. References 6

1. Document Control

1.1. Document Information

Title	Squiz Matrix CMS - Multiple Issues - Security Advisory
Document Filename	ZX Security Advisory - Squiz Matrix CMS - Multiple Vulnerabilities.docx

1.2. Revision Control

Version	Date Released	Pages Affected	Author	Description
1.0	06/12/2019	All	Stephen Shkardoon	Initial release

2. Background

2.1. Introduction

ZX Security identified several vulnerabilities the Squiz Matrix CMS that can be chained together to gain pre-authenticated remote code execution in some circumstances.

2.2. Affected versions

The issues in this advisory affect the following versions of Squiz Matrix:

- 5.5.0 prior to 5.5.0.3
- 5.5.1 prior to 5.5.1.8
- 5.5.2 prior to 5.5.2.4
- 5.5.3 prior to 5.5.3.3

2.3. Disclosure Timeline

ZX Security would like to commend the prompt response and resolution of these reported issues by the vendor.

Vendor notification:	August 09, 2019
Vendor response:	August 09, 2019
Fixed versions released:	August 29, 2019

3. Technical Findings

3.1. PHP unserialization of user input may result in remote code execution

CVE-2019-19373

When an instance of a Remote Content page exists within a Squiz Matrix CMS website, user input is passed directly and unsantized to the PHP function unserialize. In some versions of PHP (e.g. before 5.4.24), this can be leveraged into a LFI issue. If combined with arbitrary file upload with the Squiz Matrix CMS website, this leads to remote code execution.

Within packages/cms/page_templates/page_remote_content/page_remote_content.inc, the POST parameter "page_remote_content_[pageid]_sq_remote_input_file_names" is passed to unserialize. No generic unserialization gadgets were identified within the default installation, so the autoloader can be attacked instead.

There are multiple autoloaders that are enabled during the standard Squiz Matrix execution path. Of note is one found in vendor/simplesamlphp/saml2/src/_autoload.php. When given a class name that contains characters such as "." and "/", it will directly use these to include a file. This is a local file inclusion issue within the code, though is codified within PSR standards, and not normally exploitable. It should be noted however that underscores are not valid within a filename included in this method.

Using this class, we can potentially include a file simply by having PHP attempt to instantiate a class with a malicious name.

There is a second autoloader within the codebase that is not run by default: vendor/gettext/languages/src/autoloader.php. This autoloader contains the same kind of issue, however without the underscore limitation (though with other limitations, such as the class beginning with a certain string). Once again, this is part of the PSR specification, and not normally exploitable.

PHP includes within its unserialize function a check on the class name of a deserialized object to ensure it does not contain invalid characters. This means we cannot directly trigger the LFI issue using deserialize.

Instead, we can use a more standard deserialize exploitation example, where we instantiate a class that calls specific code on __destruct. Through reviewing the codebase, multiple places were found that are applicable to this case.

Consider: vendor/simplesamlphp/simplesamlphp/lib/SimpleSAML/Store/Redis.php. The destruct method of this class calls the `method_exists` function on the `$this->redis` variable, which we can control. The `method_exists` function, among many others, will trigger the autoloader with the first variable specified

(in this case, `$this->redis`, which we control). It should be noted once again that this is not the same on all versions of PHP (see references at the end of this advisory).

The last part of exploitation is a deserialize technique called "fast destruct". This allows an object to be destructed within a single deserialize call, which allows use to instantiate two classes which trigger the LFI exploit sequentially within a single request.

Putting together these steps, we can generate an unserialize payload like this:

```
$r = new SimpleSAML\Store\Redis('../../../../vendor/gettext/languages/src/autoloader');
$r2 = new SimpleSAML\Store\Redis('Gettext\Languages\../../../../x.php'); // File to include
echo serialize(array($r, $r2));
```

This gives a payload such as:

```
a:2:{i:0;O:22:"SimpleSAML\Store\Redis":1:{s:5:"redis";s:51:"../../../../vendor/gettext/languages/src/autoloader";}i:1;O:22:"S
impleSAML\Store\Redis":1:{s:5:"redis";s:220:"Gettext\Languages\../../../../data/private/assets/form_email/0008/38978/in
complete_attachments/e7b54mbvmmkfuip5tnogfter9k4ddndf81caoso02ceknl1m5ikmt1ijnn9u9bnaj861iv3tgar1e3od3
bi4l13uctm1l5uotiubrf2/38978_q1/simple_shell";}}
```

If we modify this with the fast destruct method, we get the payload:

```
a:2:{i:0;O:22:"SimpleSAML\Store\Redis":1:{s:5:"redis";s:51:"../../../../vendor/gettext/languages/src/autoloader";}i:0;O:22:"S
impleSAML\Store\Redis":1:{s:5:"redis";s:220:"Gettext\Languages\../../../../data/private/assets/form_email/0008/38978/in
complete_attachments/e7b54mbvmmkfuip5tnogfter9k4ddndf81caoso02ceknl1m5ikmt1ijnn9u9bnaj861iv3tgar1e3od3
bi4l13uctm1l5uotiubrf2/38978_q1/simple_shell";}}
```

Once we send this request to the server on a Remote Page type, we achieve LFI of a file we previously uploaded to the server, resulting in remote code execution.

3.2. Arbitrary file deletion and information disclosure in file upload form

CVE-2019-19374

When an instance of a custom form with a File Upload Field exists within a Squiz Matrix CMS website, users of the website may be able to delete arbitrary files from the server through the delete uploaded file feature. Additionally, this feature discloses the full path of files uploaded to the server, a form of information disclosure.

When a user uploads a file to a form, they can keep track of the files with the "prev_files" array, which is rendered in the HTML after a file is uploaded. This array contains the full path to each uploaded file. The relevant code can be found in:

core/assets/form/form_question_types/form_question_type_file_upload/form_question_type_file_upload.inc

An attacker can replace this path to one of their choosing, such as setting it to "data/private/conf/db.inc", and choose the delete file option. This deletes the file from the server.

4. References

For more information on the PHP unserailize fast destruct technique, see:
https://github.com/ambionics/phpggc.

For more information on exploiting the PHP autoloader, including information on exactly which PHP versions are affected, see:
https://medium.com/@ss23/php-autloading-local-file-inclusion-by-design-71aafe627877

ZX Security Limited

Level 1, 50 Manners St

Wellington, New Zealand