Jan Teriete

Objektorientiertes PHP7

Band 3: Eine Einführung in das Thema Sicherheit

Ein Webmasters Press Lernbuch

Version 2.4.0 vom 23.04.2019

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm

Inhaltsverzeichnis

Parametermanipulation

5

In dieser Lektion lernen Sie

- wie sich URL-Parameter manipulieren lassen.
- wie Blacklist- und Whitelist-Ansätze solche Manipulationen verhindern können.

5.1 Einleitung

Schauen wir uns das Template der Startseite unserer Anwendung einmal genauer an.

Beispiel

```
<div class="links">
    [ <a
        href="index.php?action=read&id=<?= $article->getId() ?>"
    >Details</a> ]
    [ <a
        href="index.php?action=edit&id=<?= $article->getId() ?>"
    >Edit</a> ]
    [ <a
        href="index.php?action=delete&id=<?= $article->getId() ?>"
    >Delete</a> ]
</div>
```

Codebeispiel 26 *templates/IndexController/indexAction.tpl.php (Ausschnitt)*

Wenn Sie sich den Ausschnitt ansehen, werden Sie feststellen, dass mehrfach eine ID als Link-Parameter ergänzt wird. Dies ist beispielsweise beim Details-Link der Fall. Doch was ist, wenn ein Besucher diese ID-Angabe manipuliert? Dabei muss es sich nicht einmal um einen böswilligen Angreifer handeln. Es kann genauso gut ein neugieriger Besucher sein, der die ID einfach einmal auf 99 erhöht. Sofern die ID nicht existiert, erhält er die Meldung `Fatal error: Call to a member function getTitle() on a non-object in (…)\templates\IndexController\readAction.tpl.php on line 3`. Diese Meldung offenbart gleich drei Details, was unsere Anwendung betrifft: Wir nutzen objektorientierte Programmierung, unsere Templates liegen im Ordner *templates*, und das Template hat einen namentlichen Zusammenhang zur aktuellen Aktion im URL-Parameter. Eine solche Schwachstelle bezeichnet man auch als unerwünschte Informationspreisgabe (engl.: ***Unplanned Information Disclosure***).

Planen Sie zu Beginn der Anwendungsentwicklung unbedingt auch Prüfroutinen für die verwendeten URL-Parameter ein.

5.2 Lösungsansätze

5.2.1 Whitelist — Variante 1

Doch wie lässt sich dieses Problem lösen und gleichzeitig die Usability unserer Anwendung etwas verbessern? Wären die Articles unveränderliche Daten und kämen niemals neue Datensätze hinzu, so könnten wir einfach die erlaubten IDs als Whitelist in unserem Code hinterlegen.

Beispiel

```
1  <?php
2
3  namespace Controllers;
4
5  class IndexController extends AbstractBase
6  {
7      // gekuerztes Beispiel
8
9      public function readAction()
10     {
11         $whitelist = [1, 2, 3, 4];
12         if (!in_array($_GET['id'], $whitelist)) {
13             die('ID nicht vorhanden!');
14         }
15
16         $em = $this->getEntityManager();
17         $article = $em
18             ->getRepository('Entities\Article')
19             ->find($_GET['id'])
20         ;
21
22         $this->addContext('article', $article);
23     }
24
25     // gekuerztes Beispiel
26 }
```

Codebeispiel 27 *src/Controllers/IndexController.php (Version 1 — ungünstige statische Whitelist)*

Man könnte die erlaubten IDs also **theoretisch** als Array in unserer Aktion notieren und dann mit der Funktion `in_array()` prüfen, ob der entsprechende URL-Parameter als Wert im Array vorhanden ist. Das Problem wären hierbei jedoch zunächst einmal die Datentypen, da selbst Zahlen im Array `$_GET` grundsätzlich als Strings vorliegen und wir in Zeile 11 die Werte im Whitelist-Array als Integers notiert haben. Die Funktion `is_array()` würde deshalb eine automatische Typwandlung vornehmen, was zu teilweise doch sehr überraschenden Ergebnissen[34] führen kann. Ein Wert wie `2test` wäre beispielsweise im Array vorhanden.[35]

Beispiel

```
1  <?php
2
3  namespace Controllers;
4
5  class IndexController extends AbstractBase
6  {
7      // gekuerztes Beispiel
8
9      public function readAction()
10     {
11         $whitelist = ['1', '2', '3', '4'];
12         if (!in_array($_GET['id'], $whitelist, true)) {
13             die('ID nicht vorhanden!');
14         }
15
16         $em = $this->getEntityManager();
17         $article = $em
```

34. http://php.net/manual/en/function.in-array.php#106319
35. Wir erinnern uns: Ein mit einer Zahl beginnender String wird in die entsprechende Zahl umgewandelt und Strings ohne Zahl am Anfang in die Zahl `0`.

```
18              ->getRepository('Entities\Article')
19              ->find($_GET['id'])
20          ;
21
22          $this->addContext('article', $article);
23      }
24
25      // gekuerztes Beispiel
26  }
```

Codebeispiel 28 *src/Controllers/IndexController.php (Version 1b — bessere statische Whitelist)*

Es wäre also sinnvoll, diese Typwandlung zu vermeiden. Zunächst sollten wir deshalb in Zeile 12 den Wert `true` für den dritten Funktionsparameter von `in_array()` angeben. Dann würden jedoch gar keine IDs mehr gefunden, da wir ja einen String in einem Whitelist-Array mit Integers suchen. Zusätzlich müssen wir deshalb die Werte im Whitelist-Array als Strings notieren. Unveränderliche datenbankbasierte Daten begegnen uns in einer Webapplikation jedoch so gut wie nie. Doch selbst wenn dies der Fall wäre, so wäre die Wartbarkeit des Codes relativ schlecht, da man im Falle einer Anpassung die Datenbankinhalte und auch immer Zeile 11 im Code aktualisieren müsste. Und eine solche Änderungsanforderung kommt meist schneller als man denkt und hofft.

5.2.2 Whitelist — Variante 2

Wir benötigen also in der Praxis eine andere Variante unserer Whitelist, bei der wir nicht die bekannten IDs im Code hinterlegen, sondern auf die vom DBMS gelieferten Daten reagieren.

Beispiel

```
1  <?php
2
3  namespace Controllers;
4
5  use Doctrine\ORM\EntityManager;
6
7  abstract class AbstractBase
8  {
9      // gekuerztes Beispiel
10
11     public function render404()
12     {
13         header('HTTP/1.0 404 Not Found');
14         die('Error 404');
15     }
16
17     // gekuerztes Beispiel
18 }
```

Codebeispiel 29 *src/Controllers/AbstractBase.php (Version 1 — Ausschnitt)*

```
1  <?php
2
3  namespace Controllers;
4
5  class IndexController extends AbstractBase
6  {
7      // gekuerztes Beispiel
8
9      public function readAction()
10     {
11         $em = $this->getEntityManager();
12         $article = $em
13             ->getRepository('Entities\Article')
14             ->find($_GET['id'])
15         ;
```

```
16
17          $article || $this->render404();
18
19          $this->addContext('article', $article);
20      }
21
22      // gekuerztes Beispiel
23  }
```

Codebeispiel 30 *src/Controllers/IndexController.php (Version 2 — flexible Whitelist)*

Sie werden sich eventuell über die Schreibweise in Zeile 17 des Controllers wundern. Wir machen uns hier eine als ***Kurzschlussauswertung*** bekannte Auswertungstechnik von PHP zunutze.

Die sogenannte Kurzschlussauswertung[36] (engl.: Short-Circuit Evaluation) ist ein Spezialfall der Lazy Evaluation[37] (dt.: faule/bequeme Auswertung), der auf logische Ausdrücke ausgerichtet ist. Im Allgemeinen steht das Gesamtergebnis eines booleschen Ausdrucks erst nach der Auswertung aller Teilausdrücke fest. Eine Kurzschlussauswertung ermöglicht das vorzeitige Abbrechen einer solchen Auswertung, sobald das Gesamtergebnis durch einen Teilausdruck eindeutig bestimmt ist. Logische Ausdrücke werden von links nach rechts ausgewertet, `false &&` (irgendwas) ist immer `false` und `true ||` (irgendwas) ist immer `true`.

In beiden Fällen erfolgt deshalb in PHP eine Kurzschlussauswertung. Dies kann beispielsweise eine unnötige rechenintensive Auswertung komplexerer Teilausdrücke verhindern (Stichwort Zeitersparnis). Auch lassen sich so Ausführungsfehler vermeiden, wenn beispielsweise die Auswertung des zweiten Teilausdrucks nur möglich ist, wenn der erste Teilausdruck zu `true` ausgewertet werden kann.

Der hintere Teil der Oder-Verknüpfung in Zeile 17 wird also nur ausgewertet, wenn der Finder anhand der ID keinen Datensatz findet und `$article` deswegen `null` enthält (entspricht `false`). Nur dann steht das Gesamtergebnis des Ausdrucks noch nicht fest und die Methode muss ausgeführt werden, womit dann die Ausgabe der 404-Meldung erfolgt.

Haben Sie gedacht, dass der ID-Parameter nun sicher gegen sämtliche Manipulationsversuche ist? Weit gefehlt, denn wenn man den Parameter komplett entfernt und nur die Aktionsangabe übrig lässt, so erhält man eine Reihe von informativen PHP-Meldungen. Beispielsweise erfährt man durch diese, dass Doctrine mit der Klasse `Entities\Article` verwendet wird. Dieses Problem haben wir bereits in »Band 1: Grundlagen der OOP« beim Thema MVC für den Aktions-Parameter gelöst. Wir haben dabei den Null-Coalesce-Operator verwendet. Ich persönlich bin hierzu ehrlich gesagt zu faul und kombiniere ein simples ***Type Casting*** mit der Deaktivierung der Fehleranzeige.

Beispiel

```
1  <?php
2
3  namespace Controllers;
4
5  class IndexController extends AbstractBase
6  {
7      // gekuerztes Beispiel
8
9      public function readAction()
10     {
11         $em = $this->getEntityManager();
12         $article = $em
```

36. http://php.net/manual/en/language.operators.logical.php
37. https://de.wikipedia.org/wiki/Lazy_Evaluation

```
13              ->getRepository('Entities\Article')
14              ->find((int)$_GET['id'])
15          ;
16
17          $article || $this->render404();
18
19          $this->addContext('article', $article);
20      }
21
22      // gekuerztes Beispiel
23  }
```

Codebeispiel 31 *src/Controllers/IndexController.php (Version 3)*

Was passiert nun? In Zeile 14 erzwingen wir den Datentyp Integer. Fehlt der URL-Parameter komplett, so wird hierdurch die ID `0` erzwungen. Diese ID gibt es aber bei keinem normalen Datensatz, und somit greift die Whitelist in Zeile 17. Wir erhalten natürlich trotzdem noch die Hinweismeldung für den **nicht** vorhandenen Array-Index, aber nur sofern wir die Fehleranzeige (noch) nicht deaktiviert haben und überhaupt jemand die URL-Parameter manipuliert. Denken Sie also unbedingt daran, im **produktiven Umfeld** den `debug_mode` in der *default-config.php* zu deaktivieren.

Trauen Sie niemals den Eingaben und Angaben eines Benutzers. Hierzu zählen beispielsweise URL-Parameter und sämtliche Formulardaten (auch die von versteckten Feldern und Radiobuttons). URL-Parameter lassen sich problemlos direkt in der URL-Angabe manipulieren. Bei der Manipulation von Formulardaten helfen im Firefox beispielsweise die integrierten Developer-Tools[38] (insbesondere der HTML-Inspektor).

5.3 Dateiangaben als Parameter

Betrachten wir noch einmal den grundsätzlichen Ablauf in unserer Anwendung:

1. Ein Benutzer ruft die Anwendung über den Front-Controller (mit oder ohne URL-Parameter) auf.
2. Bei einem korrekten Controller-Parameter in der URL wird eine Controller-Klasse instanziiert und die Methode `AbstractBase#run()` in diesem Objekt aufgerufen.
3. Der Aktions-Parameter landet dann im Attribut `$template` dieser Instanz, wobei er unter anderem mit dem Suffix `Action` und der Dateiendung *.tpl.php* ergänzt wird.
4. Sofern der Aktions-Parameter einer existierenden Methode in der Controller-Klasse zugeordnet werden kann, wird diese aufgerufen.
5. Durch die Methode `AbstractBase#render()` wird abschließend das Template *layout.tpl.php* per `require_once` eingebunden.
6. Dieses Layout-Template bindet per `require` wiederum das eigentliche Template der Aktion ein, wobei sich Letzteres in einem controllerspezifischen Unterordner von *templates* befindet.

Bei dem ganzen Ablauf gehen wir davon aus, dass die Template-Anzeige nur in der vorgesehenen Art und Weise über die beiden URL-Parameter manipuliert werden kann, da wir ja den Pfad und die Dateiendung fest vorgeben. Vor PHP 5.3.4 war dies jedoch leider nicht der Fall, da weitergehende Manipulationen mit sogenannten ***Null Bytes*** möglich waren. Hierdurch hätte der PHP-Interpreter beispielsweise bei uns das Suffix `Action` und die Dateiendung *.tpl.php* ignoriert[39].

38. https://developer.mozilla.org/en-US/docs/Tools/Tools_Toolbox
39. http://php.net/de/security.filesystem.nullbytes

Beispiel

```
1  <?php
2
3  namespace Controllers;
4
5  use Doctrine\ORM\EntityManager;
6
7  abstract class AbstractBase
8  {
9      // gekuerztes Beispiel
10
11     public function run($action)
12     {
13         $this->addContext('action', $action);
14
15         $methodName = $action; // . 'Action';
16         $this->setTemplate($methodName);
17
18         if (method_exists($this, $methodName)) {
19             $this->$methodName();
20         } else {
21             //$this->render404();
22         }
23
24         $this->render();
25     }
26
27     public function render404()
28     {
29         header('HTTP/1.0 404 Not Found');
30         die('Error 404');
31     }
32
33     // gekuerztes Beispiel
34
35     protected function setTemplate($template, $controller = null)
36     {
37         if (empty($controller)) {
38             $controller = $this->getControllerShortName();
39         }
40
41         $this->template = $controller . '/' . $template; // . '.tpl.php';
42     }
43
44     // gekuerztes Beispiel
45
46     protected function render()
47     {
48         extract($this->context);
49
50         $message = $this->getMessage(); // Get flash message
51         $template = $this->getTemplate();
52
53         require_once $this->basePath . '/templates/layout.tpl.php';
54     }
55 }
```

Codebeispiel 32 *src/Controllers/AbstractBase.php (Version 2 — Ausschnitte)*

Gehen wir für einen kleinen Test davon aus, dass wir bei einer fehlenden Methode kommentarlos den restlichen Code der Controller-Klasse ausführen. Wir kommentieren Zeile 21 deshalb einfach aus. Manipulieren Sie testweise den Aktions-Parameter Ihrer Anwendung mit der Angabe `action=../../composer.json%00`. Wird dies **nicht** mit einem Fatal error quittiert, so ist Ihr PHP-Interpreter noch anfällig für eine ***Remote File Inclusion*** mittels ***Null Byte Termination***. Dies sollte heutzutage jedoch eigentlich nicht mehr der Fall sein.

Eine ***Remote File Inclusion*** ist eine Sicherheitslücke in einer Webanwendung, die es einem Angreifer erlaubt, unkontrolliert Code in diese einzuschleusen und auszuführen. Eine solche Lücke entsteht bei einer PHP-Anwendung, wenn Benutzereingaben unzureichend geprüft bei einem require(_once) bzw. include(_once) verwendet werden. Beispielsweise würde ein `../` bzw. ein `/` in einer solchen Benutzereingabe einen Verzeichniswechsel ermöglichen. Schlimmstenfalls kann über eine solche ***Remote File Inclusion*** sogar Code von einem fremden Webserver[40] ausgeführt werden. Als Absicherung bietet es sich beispielsweise an, in einer solchen Benutzereingabe durch eine Whitelist nur Buchstaben zu erlauben. Da nun keine Sonderzeichen mehr möglich sind, sollte auch jeglicher Versuch eines Verzeichniswechsels unterbunden werden.

Bedenken Sie bei der Absicherung einer Anwendung unbedingt, dass böswilliger PHP-Code durch Benutzer-Uploads auf Ihren Webserver gelangen kann. Die *.htaccess*-Dateien mit Whitelist-Ansatz aus Abschnitt 4.3 beschränken zwar die Ausführung von PHP-Dateien auf vorgegebene Dateinamen, doch betrifft dies nur eine direkt im Browser aufgerufene Datei. Über eine ***Remote File Inclusion*** könnte diese wiederum eine beliebige weitere Datei einbinden und ausführen.

Um bei dem obigen Beispiel auch heutzutage noch ein Resultat sehen zu können, emulieren wir das Problem mit den Null Bytes, indem wir das Suffix (Zeile 15) und die Dateiendung (Zeile 41) nicht mehr ergänzen. Mit der Angabe `action=../../composer.json` erhalten wir nun eine Ausgabe der *composer.json*.

Da man, sofern möglich, mehrere Verteidigungslinien benutzen sollte, sollten Sie sich nicht nur auf die gefixte PHP-Version verlassen. Brechen Sie bei einer fehlenden Methode für die Aktion immer die Verarbeitung komplett ab (z. B. mit unserer `render404`-Methode). Diese Umsetzung einer Whitelist nutzen Sie theoretisch seit »Band 1: Grundlagen der OOP«, wo sie im Rahmen einer Aufgabe erarbeitet werden sollte. Zudem sollten Angaben wie Ordner, Suffixe und Dateiendungen fest in Ihrem Code verankert sein und nicht über URL-Parameter übergeben werden können.

Beispiel

```
1  <?php
2
3  require_once 'inc/functions.inc.php';
4  require_once 'inc/helper.inc.php';
5
6  require_once 'inc/bootstrap.inc.php';
7
8  // Session needed for flash messages
9  session_start();
10
11 // Path to our index.php
12 $basePath = dirname(__FILE__);
13
14 $controller = $_GET['controller'] ?? 'index';
15 $controller = preg_replace('/[^a-z]/', '', $controller);
16
17 $action = $_GET['action'] ?? 'index';
18 $action = preg_replace('/[^a-z]/', '', $action);
19
20 $controllerNamespace = 'Controllers\\';
21 $controllerName = $controllerNamespace . ucfirst($controller) . 'Controller';
22
23 if (class_exists($controllerName)) {
24     $requestController = new $controllerName($basePath, $em);
```

40. http://php.net/manual/de/features.remote-files.php

```
25      $requestController->run($action);
26  } else {
27      $requestController = new Controllers\IndexController($basePath, $em);
28      $requestController->render404();
29  }
```

Codebeispiel 33 *index.php*

Als zusätzliche Verteidigungslinie erlauben wir durch eine ***Filterung*** in Zeile 15 und 18 nur noch Kleinbuchstaben in den Variablen `$controller` und `$action`. Alle Zeichen, die dieser Whitelist nicht entsprechen (wie beispielsweise der von uns zum Verzeichniswechsel verwendete Slash `/`), werden einfach gelöscht. Somit sind die Variablen nun wirklich komplett abgesichert und `$action` kann auch problemlos in unseren Templates ausgegeben werden. Bei `$controller` ist eine solche Template-Ausgabe derzeit übrigens nicht möglich, da wir die Variable nicht an unsere Templates weitergeben.

Verwenden Sie in Prüfroutinen von Include- und Require-Operationen möglichst immer einen Whitelist-Ansatz, um eine ***Parametermanipulation*** zu verhindern.

5.4 Testen Sie Ihr Wissen!

1. Sind zwei Bedingungen durch `||` verknüpft und die erste Bedingung wird als `true` ausgewertet, so wird die zweite Bedingung nicht ausgewertet. Wie nennt man diese Auswertungstechnik?

2. Sind zwei Bedingungen mittels `&&` verknüpft und die erste Bedingung wird als `false` ausgewertet, erfolgt dann in PHP noch eine Auswertung der zweiten Bedingung?

3. In der PHP-Anwendung eines Kollegen wurde eine Sicherheitslücke gefunden. Sie ist anfällig für eine ***Remote File Inclusion***. Was hat der Kollege falsch gemacht bzw. unterlassen?

4. Bei welchem Parameter muss man bei der Funktion **in_array()** den Boolean true übergeben, um eine automatische Typwandlung zu verhindern?

   Bitte ankreuzen:

   ☐ erster Parameter

   ☐ zweiter Parameter

   ☐ dritter Parameter

   ☐ vierter Parameter

5. Gegeben sei nachfolgender PHP-Code. Was wird ausgegeben?

```
if (in_array('4zehn', [1, 6, 4])) {
    echo 'a';
}

if (in_array('0zehn', [false, '0', ''])) {
    echo 'b';
}

if (in_array('0', [true, false, null])) {
    echo 'c';
}
```

   Antwort:

6. Die Funktion **test** wird mit nachfolgender Codezeile aufgerufen.

```
$article && test($article);
```

Welche Aussage ist korrekt?

Bitte ankreuzen:

- ☐ Die Funktion **test** wird nur aufgerufen, wenn $article zu true ausgewertet werden kann.
- ☐ Die Funktion **test** wird nur aufgerufen, wenn $article zu false ausgewertet werden kann.
- ☐ Die Funktion **test** wird nur aufgerufen, wenn $article den Wert null hat.
- ☐ Die Funktion **test** wird unabhängig vom Wert von $article immer ausgerufen.

7. Gegeben sei nachfolgender PHP-Code. Was wird ausgegeben?

```
function holeZahlen($index)
{
    return [0, 4][$index];
}

holeZahlen(0) && die('!lazy1');
holeZahlen(1) && die('!lazy2');
```

Antwort:

8. Gegeben sei nachfolgender PHP-Code. Was wird ausgegeben?

```
function holeWerte($index)
{
    return ['0', []][$index];
}

holeWerte(0) || die('!lazy3');
holeWerte(1) || die('!lazy4');
```

Antwort:

5.5 Übungen

Übung 7:

Integrieren Sie die zeichenbasierte Whitelist für `$controller` und `$action` im Front-Controller.

Übung 8:

Schützen Sie den Newsticker mit der vorgestellten Error404-Whitelist inklusive eines vorherigen Type Castings gegen eine Manipulation von ID-Parametern. Dies ist überall dort sinnvoll, wo ein einzelner Datensatz anhand seiner ID ausgelesen werden soll und der Wert der ID aus URL-Parametern oder aus Formulardaten stammt. Im Zusammenhang mit der Methode `Article#addTag()` ist dies jedoch meiner Meinung nach unnötig, da hier eine Manipulation bereits durch die Typdeklaration verhindert wird. Allerdings schadet es nichts, wenn Sie mittels `find((int)$id)` einen Integer für die IDs erzwingen.