International Journal of Computers and Applications

ISSN: 1206-212X (Print) 1925-7074 (Online) Journal homepage: http://www.tandfonline.com/loi/tjca20

Security test MOODLE: a penetration testing case study

Akalanka Karunarathne Mudiyanselage & Lei Pan

To cite this article: Akalanka Karunarathne Mudiyanselage & Lei Pan (2017): Security test MOODLE: a penetration testing case study, International Journal of Computers and Applications

To link to this article: http://dx.doi.org/10.1080/1206212X.2017.1396413

Published online: 13 Nov 2017.

INTERNATIONAL JOURNAL OF COMPUTERS AND APPLICATIONS, 2017
https://doi.org/10.1080/1206212X.2017.1396413

Security test MOODLE: a penetration testing case study

Akalanka Karunarathne Mudiyanselage[a] and Lei Pan[b]

[a]EY, Melbourne, Australia; [b]Institut School of IT, Deakin University, Geelong, Australia

ABSTRACT
Moodle project http://moodle.org is one of the most widely used web application packages for delivering teaching materials in universities and colleges. Despite its popularity and high level of acceptance of teachers and decision-makers, the security aspects of Moodle has not been well-mentioned in publications; to our best knowledge, no active research has been conducted to assess the level of security assurance of Moodle. Because of this lack of Knowledge, many Moodle sites were, have been or are exploited in the following manner – propriety teaching materials were stolen, instructors or administrator's credentials were compromised, student results were changes and so on. This paper demonstrates a security testing case for identifying security vulnerabilities of Moodle. Using automated, manual source code review and web application penetration testing we have demonstrated a sound PHP-based security assessment methodology for Moodle. As a result, we have identified nine security vulnerabilities from Moodle 2.6. The description and security analysis of these vulnerabilities are provided in details. In order to utilize this framework, the tester should have advanced technical skills in operating a source code scanner to differentiate the false positives the scanning results, advanced source code review skills to identify application logic related vulnerabilities and advanced web application penetration testing skills to validate the results from the automated/manual source code review and technical findings that were missed during the source code review. Readers of this paper are encouraged to use our methodology to perform security assessment for their versions of Moodle and to extend the features/functionalities of the testing methodology to further expand the test coverage.

ARTICLE HISTORY
Received 18 January 2017
Accepted 20 August 2017

KEYWORDS
Penetration testing; cyber security; Moodle; vulnerability; security analysis

1. Introduction

Modular Object-Oriented Dynamic Learning Environment, or Moodle, is a free software in the e-learning platform. Due to the fast-growing demands of methodologies and technologies there is a higher demand for e-Learning. According to [1],

> eLearning refers to the exploration and use of a broad range of information and communication technologies to provide new learning environments, that may be interactive or accessed online from home or within the community. The inclusion of e-Learning give students new and valuable learning experiences.

Moodle is used in many institutions and industries as their e-learning platform; the following is a list of industries that commonly use Moodle: Universities, high schools, primary schools, government departments, health care organizations, military organizations, airlines, oil companies, independent educators and so on.

Exploitation of a weakness (e.g. incorrect implementation of a function in the web application) that allows a malicious user to compromise the web application's confidentiality, integrity, and availability are defined as vulnerability.

We have used the following list of top 10 vulnerabilities identified by the Open Web Application Security Project (OWASP) [2] as the benchmark for our security study of Moodle and also to categorize the issues that we have found within the Moodle framework:

(1) Injection – The most common injection flaw that has been around for decades is SQL injections which occur when the malicious user supplied data are sent to an interpreter as a query.
(2) Broken Authentication and Session Management – Weak implementation of the authentication and session handling functionalities could be exploited by the attackers to retrieve passwords or session tokens. By further exploiting these weaknesses, attackers can also take control of the accounts.
(3) Cross-Site Scripting Vulnerability (XSS) – The XSS vulnerability occurs when the application sends the malicious data into the application without proper validation of the user controllable data.
(4) Insecure Direct Object Reference – Most developers expose references to the internal implementation object (e.g. file, root directory path, database

CONTACT Lei Pan l.pan@deakin.edu.au

keys). Without any access control checks, attackers can manipulate these objects to access unauthorized data.

(5) Security Misconfiguration – It is necessary that the secure configure is defined and deployed for the application, web server, database server, frameworks, application server, and platform. The secure settings must never be set to default settings. In addition, it is necessary that the software should always be up to date.
(6) Sensitive Data Exposure – All the sensitive data such as credit card numbers, credentials, TAX identifications must always be encrypted with strong encryption methodologies.
(7) Missing Function Level Access Control – If the application fails to verify the requests, then the attackers can forge requests to access functionalities without proper authorization.
(8) Cross-Site Request Forgery (CSRF) – CSRF is an attack that allow the malicious users to enforce the victim user to perform arbitrary actions without their knowledge. For instance, attackers could change an individual's email address without their knowledge.
(9) Using Components with Known Vulnerabilities – Since the frameworks, libraries, and software modules run with full privileges, If attackers were able to exploit these software modules, they could potentially take full control over the web server.
(10) Unvalidated Redirects and Forwards – Almost all the web applications use redirections and forwarding of the users to other pages within the scope of the web application. However, if the application fails to validate the redirection and forwarding of the user's URL and file paths, then the attackers could use this vulnerability to divert victim users to phishing and/or malware sites.

Our research question is to find a simple but effective method to conduct security testing on Moodle. We chose Moodle 2.6 as the testing platform due to the fact that it is an obsolete version so that our findings will not affect mainstream Moodle users.

In this paper, we propose a testing framework which uses both automated vulnerability scanners and manual test to find security vulnerabilities. This method works well so that nine security vulnerabilities were found in Moodle 2.6.

The rest of this paper is organized as follows: Section 2 presents related work on security testing. Section 3 describes our security testing methodology. Section 4 provides the detailed descriptions and security analysis of the nine found vulnerabilities. And this paper is concluded in Section 5.

2. Related work

2.1. Source code analysis

Source code analysis helps to identify vulnerabilities because it analyses the entire source code of the Moodle framework. However, the source code analysis could be conducted either statically or dynamically.

The static approach was used to analyze the entire source and highlight the entry points to the web application, while on the other hand the dynamic approach was used to filter out the critical functions that may potentially be vulnerable due their weak implementation. Hence, source code review is a well-accepted method to identify vulnerabilities within the Moodle web application.

Dynamic analysis allows filtering of critical functions within the source code of the web application, which are potentially vulnerable. This approach requires identifying the critical functions of the web application by studying how the code behaves.

Using the dynamic source code review approach we were able to find vulnerability in the Moodle application forum. Where the `$post->message` is, the user supplied parameter that is then echoed back onto the forum page. We were able to find this vulnerability by subsequently testing how the user-supplied values are handled by the web application. Therefore, we can confirm that upon exploitation of this vulnerability the attackers could redirect the user to malicious web sites. Therefore, the dynamic approach has proven accurate and thorough. However, the dynamic approach has consumed a considerable amount of time when detecting vulnerabilities within the web application.

Using the static analysis, we can highlight the entry points to the web application, any known weak functions, and weak implementations.

2.1.1. Entry points

The entry points are like gates to a web application, which allows users to enter certain values. There are two main entry points to a web application:

(1) User input forms; and/or
(2) User controllable parameters.

User input forms fields are expected to be tampered by the users therefore the application developers must put more effort into the validation and the sanitization of those user supplied values and the user controllable fields are the values that are submitted from the client side. The following is a list of common entry points to a PHP-based web application that was compiled according to [3]:

- `$_GET`
- `$_POST`
- `$_REQUEST`
- `$_COOKIE`

2.1.2. Potentially vulnerable PHP functions

PHP is a very popular programming language that is used by the Moodle developers for their server side scripting. According to [3], the following is a list of potentially dangerous PHP functions that are often used to find vulnerable implementation of the application:

```
include
include_once
require
require_once
die
mysql_query
eval
exec
system
```

There are two ways of conducting a static source code analysis, which allows us to determine vulnerable implementations within the Moodle framework. However, conducting a static analysis requires the analysis to have prior knowledge about the programming language and the functions that it may use. We have used the static analysis to highlight the potential entry points to the Moodle web application. Following that, we used the dynamic source code analysis to filter the critical functions, which may potentially be vulnerable.

In addition, we have used the concept of Pixy. Pixy is a source code analysis tool that targets detecting cross-site scripting vulnerabilities in PHP scripts [4]. Using this approach we can highlight the taint-style vulnerabilities.

2.2. Web application penetration testing

Using the web application penetration testing approach we were able to validate the identified findings from the static and dynamic analysis phases. The web application penetration testing is a widely used method for evaluating the security of the web application.

There are two main approaches to conducting a web application penetration testing: Black-box testing and white-box testing.

In black box penetration testing, the testers have no knowledge about the test target. They have to figure out the loopholes of the system on their own from scratch. This is similar to the blind test strategy in which there is a stimulation of the actions and procedures of a real attacker who has no information concerning the test target [5].

The white box penetration testing is the opposite of the black box approach. In the white box penetration testing approach, we were able to validate the findings that were identified during the static and dynamic analysis. For instance, using the white box penetration testing approach we were able to confirm the sample vulnerability.

Penetration testing has proven to be an efficient method to audit the security of an application [6]. Therefore, using the web application penetration testing we were able to validate the findings from the static and dynamic analysis.

2.3. Vulnerability identification using automated tools

According to [7], RIPS detects sensitive sinks that can be tainted by user input during the program flow. During the static analysis of the web application source code, it is necessary to identify user supplied data also known as tainted data and detecting if the web application is allowing those tainted data to be reached to vulnerable parts of the web application. If any of these user-tainted data reaches a sensitive sink then it is called taint-style vulnerabilities.

The RIPS analyzer searches for taint-style vulnerabilities in an effective manner. However, it does provide a large number of false positive results, which consumes a large amount of time to differentiate the false positive results from the valid result.

3. Our security testing methodology

3.1. Experiment environment

Figure 1 shows the testing environment for the Moodle security testing. We use the three physical machines for testing. The machines are dedicated with the following purposes:

- The MacBook runs two operating systems – the first OS with IP address 10.0.0.2 is its native OSX hosting MySQL database services for Moodle; and the other with IP address 10.0.0.17 is a Kali Linux virtual machine for launching security attacks.
- The tower PC with IP address 10.0.0.26 runs Windows Server 2008 hosting an LDAP service for Moodle authentication.
- The Thinkpad laptop with IP address 10.0.0.18 runs Ubuntu Server hosting web services which supports Moodle's front end. From the end-user's perspective, this is the entry point to logon to their Moodle site.

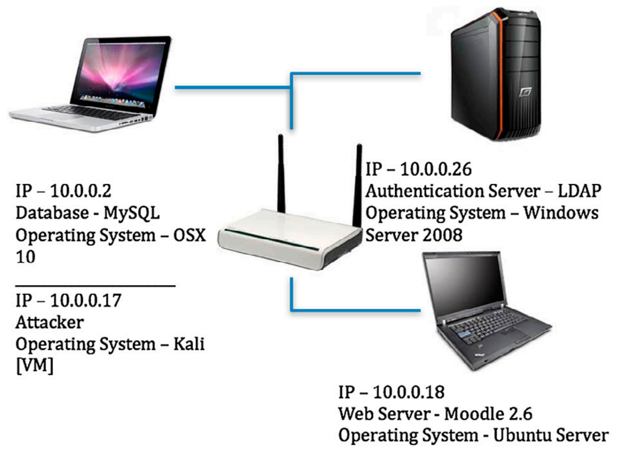

IP – 10.0.0.2
Database - MySQL
Operating System – OSX 10
IP – 10.0.0.17
Attacker
Operating System – Kali [VM]
IP – 10.0.0.26
Authentication Server – LDAP
Operating System – Windows Server 2008
IP – 10.0.0.18
Web Server - Moodle 2.6
Operating System - Ubuntu Server

Figure 1. Security Testing Environment for Moodle – Four operating systems are installed on three physical machines located in the same LAN.

3.2. Scanning result collection

The source code analysis is one of the methods used throughout the security study of the Moodle framework that has proven successful to many who have done research in similar fields.

For instance, the research conducted by [4] for the 'Pixy: static tool for detecting vulnerabilities in web application' supported during organizing the structure for this source code analysis.

Our approach consists of three steps shown in Figure 2 – using an automated tool (source code analyzer), using the PHP static source code analysis tool (RIPS), and manual source code review. The manual source code review approach is useful to reduce the false positive and false negative results that are identified during the automated scans.

Firstly, the automated tool (source code analyzer) allows me to identify known vulnerable implementation of the PHP source code. As a result, this tool will produce a csv file that lists the vulnerable functions used in PHP. In addition, it can be customized to identify the database queries within the server side code. Identification of these potentially vulnerable PHP functions and database queries must be further explored in order to determine their secure implementation. This can be achieved by conducting a dynamic code analysis, which allows the analyst to review the potentially vulnerable function and its dependencies. For example, Figure 3 shows the search results of security vulnerabilities detected by the source code analyzer tool.

Suppose that we follow the PHP function `echo` from the line 29 in Figure 3, we will locate the corresponding source code in line number 359 of `moodle26/admin/index.php` as follows:

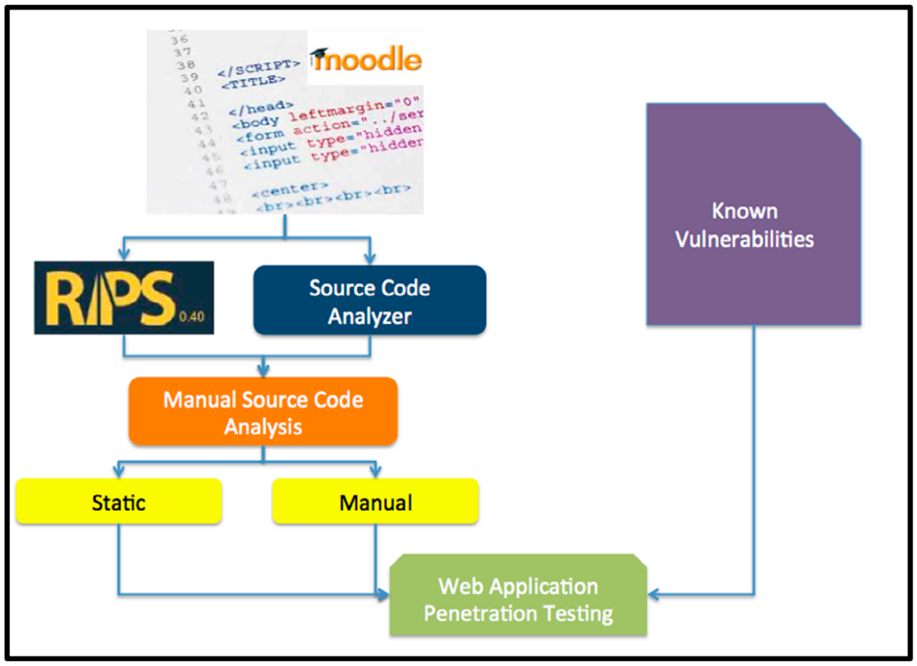
```
echo $OUTPUT->confirm(get_string
            ('alreadyloggedin'.
            'error'. fullname($USER)),
              $logout,  $continue);
```

The above code could be attacked due to the fact that the victim user may execute something malicious altered by the attacker through the values of the variables. The code analyzer is an excellent tool that is only good if 'you have known the knowledge and experience to use it'. This tool essentially searches for potentially vulnerable function that the user assigns it to search for. Hence, this tool heavily depends on the tester's knowledge in PHP security coding.

Secondly, the PHP static source code analysis tool RIPS [8] is used to scan the entire Moodle framework to identify weak implementation of the PHP functions within the Moodle framework. RIPS is a tool that is configured to with a pre-configured with set of rules that is used to identify any known vulnerable/weak implementation of the PHP functions. RIPS can also list all the dependencies of the functions within the application and provide a visual structure of the source code repository. We will use RIPS to scan the entire Moodle framework to identify all possible weak implementation. However, the downside of RIPS is that it returns a large number of false positives that is time consuming to verify.

The above method has proven to be effective when detecting the vulnerabilities within web applications. However, it has time constraints because manual source code analysis could potentially take a long time depending on the size of the web application. Web application penetration testing is used to assure the security of web applications and its services, so that the identified security weaknesses can be evaluated and fixed before they get exposed. The motivation for implementing the web application penetration-testing phase within the security testing of Moodle was to test and confirm the potential weak implementations discovered during the source analysis and known Moodle exploits. The web application penetration for Moodle followed the white-box testing where the internal working (that is, logic and the code structure) of the framework is fully visible to the security tester. The web application penetration testing is the last hop in the testing model to confirm that the vulnerability actual exits within Moodle. After the web application penetration testing phase there is the need for final verification, which is used to assigning the risk factor to these weaknesses.

Finally, the manual source code review is designed to reiterate through the entire Moodle source files in order to confirm all vulnerabilities that were not identified by the automated tools (source code analyzer and the RIPS)

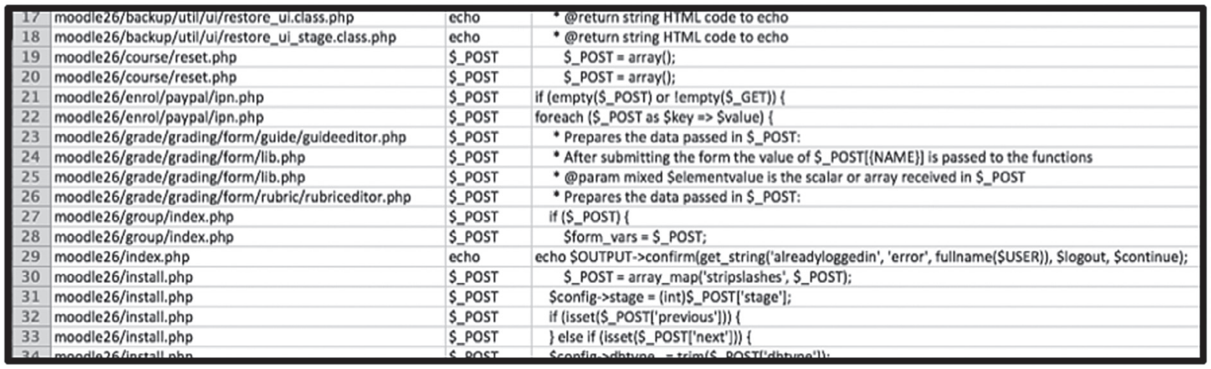


```
</SCRIPT>
<TITLE>

</head>
<body leftmargin="0"
<form action="../ser
<input type="hidden
<input type="hidden

<center>
<br><br><br><br>
```

Figure 2. Our proposed method to analyze and discover vulnerabilities of the Moodle framework.



17	moodle26/backup/util/ui/restore_ui.class.php	echo	* @return string HTML code to echo
18	moodle26/backup/util/ui/restore_ui_stage.class.php	echo	* @return string HTML code to echo
19	moodle26/course/reset.php	$_POST	$_POST = array();
20	moodle26/course/reset.php	$_POST	$_POST = array();
21	moodle26/enrol/paypal/ipn.php	$_POST	if (empty($_POST) or !empty($_GET)) {
22	moodle26/enrol/paypal/ipn.php	$_POST	foreach ($_POST as $key => $value) {
23	moodle26/grade/grading/form/guide/guideeditor.php	$_POST	* Prepares the data passed in $_POST:
24	moodle26/grade/grading/form/lib.php	$_POST	* After submitting the form the value of $_POST[{NAME}] is passed to the functions
25	moodle26/grade/grading/form/lib.php	$_POST	* @param mixed $elementvalue is the scalar or array received in $_POST
26	moodle26/grade/grading/form/rubric/rubriceditor.php	$_POST	* Prepares the data passed in $_POST:
27	moodle26/group/index.php	$_POST	if ($_POST) {
28	moodle26/group/index.php	$_POST	$form_vars = $_POST;
29	moodle26/index.php	echo	echo $OUTPUT->confirm(get_string('alreadyloggedin', 'error', fullname($USER)), $logout, $continue);
30	moodle26/install.php	$_POST	$_POST = array_map('stripslashes', $_POST);
31	moodle26/install.php	$_POST	$config->stage = (int)$_POST['stage'];
32	moodle26/install.php	$_POST	if (isset($_POST['previous'])) {
33	moodle26/install.php	$_POST	} else if (isset($_POST['next'])) {
34	moodle26/install.php	$_POST	$config->dbtype = trim($_POST['dbtype']);

Figure 3. Search results from a source code analyzer tool, where it searches for potentially vulnerable PHP functions within the Moodle source code.

and, more importantly, to confirm the validity of the results generated by the automated tools.

4. Testing – results and security risk analysis

This Section lists all results we collected by testing Moodle 2.6. For the results from the RIPS, we double check which result is vulnerable. Within each finding we provide a detailed description of the issue within the Moodle and its impact to the Moodle framework by conducting a threat analysis according to different user groups.

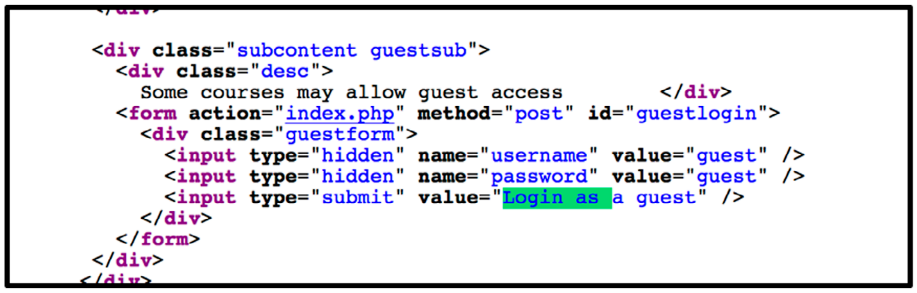
4.1. Low risk: hardcoded guest account username and password

The application contains a default guest user account and this guest user account does not have any privileges as default however, the guest is granted access to some functions (e.g. search for courses, etc.) with the Moodle application. It was noted that the guest users credentials are hardcoded to the application source code.

The guest user's username 'guest' and password 'guest' is hardcoded to the login page at `login/index.php`, as shown in Figure 4.

Some functions within the Moodle framework can be accessible to the guest user as default. For example, the following pages can be accessible by guest users: calendar, course search, and tags and forum posts.

This is considered a low-level risk because the application does not allow critical functions to the guest user. However, the administrator has the ability to grant course access guest user. Furthermore, some functions (e.g. forum posts) within Moodle could be useful to attackers for further exploiting the system.



```
    <div class="subcontent guestsub">
      <div class="desc">
        Some courses may allow guest access        </div>
      <form action="index.php" method="post" id="guestlogin">
        <div class="guestform">
          <input type="hidden" name="username" value="guest" />
          <input type="hidden" name="password" value="guest" />
          <input type="submit" value="Login as a guest" />
        </div>
      </form>
    </div>
```

Figure 4. The guest users username 'guest' and its password 'guest' is hard coded to the application login page.

4.2. *Low risk: AUTOCOMPLETE is not explicitly disabled*

The AUTOCOMPLETE feature of the password field in the Moodle login page at `login/index.php` is not explicitly disabled, as shown in the bottom line of Figure 5.

The AUTOCOMPLE feature is used to keep track of the password field information that has recently been submitted to the login page. For example, the above screenshot shows that the username and password fields are not explicitly disabled causing the fact that the username and password information can be stored in the browsers' cache. When the victim's cache is compromised, the stored username and password can be retrieved.

The application users are students and teacher who access the application using a shared computer can allow the attackers to access the browser cache to retrieve the username and password which allows them to compromise the victims' account.

4.3. *Medium risk: weak input validation*

The Moodle 2.6 web application does not implement best practice during the input validation of the user controllable data. The term user controllable means all the parameter values that could potentially be controlled by the user using an intercepting tool such as Burp Interceptor or Paros to conduct a parameter tampering or FUZZ attack.

The Moodle forum page does not validate the *$data* parameter for potentially malicious payload, as a result we were able to inject a payload into the forum that would allow us to conduct a phishing attack (Figure 6).

This vulnerability exists because the *$data->message* parameter is not properly sanitized which allows certain payloads can be inserted an echoed back onto the Moodle message page at `message/index.php`.The following PHP code shows that the un-sanitized message parameter being echoed back onto the message page at `message/index.php`:

```
echo $OUTPUT -> box_start('message');
```

Using an intercepting proxy, we were able to tamper the message parameter and inject malicious code, which allows us to potentially conduct phishing attacks against the Moodle users. The payload that we have chosen allows us to conduct a phishing style attack, which could potentially steal the users' username and password.

This vulnerability is of medium level risk. The ability to insert malicious payload onto the application allows the attackers conduct malicious actions against other Moodle users:

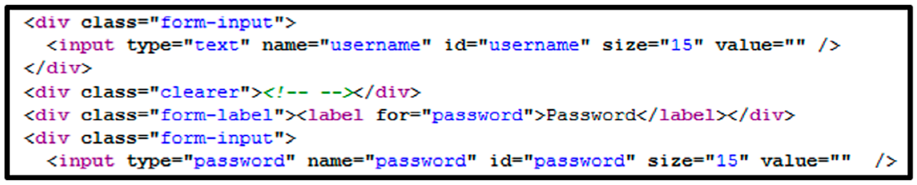
- Student user: The attackers is a potential student that is trying to retrieve the credentials of another student in order to hijack their user account that could potentially allow them to breach the confidentiality, integrity, and the availability of the protected data. For instance, the attacker could be another student that is attempting to steal the victim students completed assignment or to delete the submitted assignment in order to sabotage.
- Lecturer user: The attacker could be a student attempting to get access to their lecturers' account, which could allow them to gain access to the protected resources such as upcoming assignments, exam papers, etc.
- Admin user: The potential attacker can be anyone that has the motivation to sabotage the organization and its business operations.

4.4. *Medium risk: brute force-able login interface*

The Moodle 2.6 login interface is vulnerable to brute force attack.

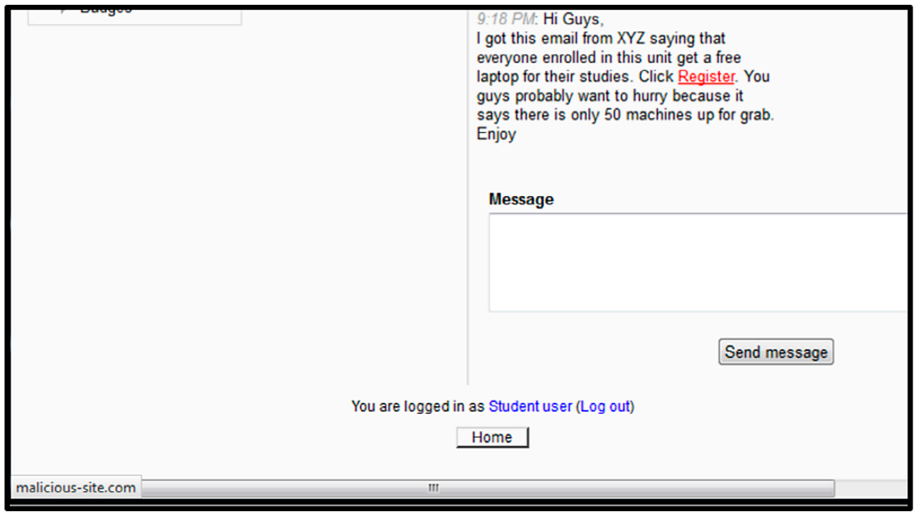
Brute force attack is the systematic approach to enumerate all possible candidates of the login page. The typical login pages contain a username and password field where the brute force attack would potentially enumerate the username or the password to potentially get access to a certain user account.

There are two main type of brute force attack: Horizontal brute force and Vertical brute force. Using a list of



```
<div class="form-input">
  <input type="text" name="username" id="username" size="15" value="" />
</div>
<div class="clearer"><!-- --></div>
<div class="form-label"><label for="password">Password</label></div>
<div class="form-input">
  <input type="password" name="password" id="password" size="15" value=""  />
```

Figure 5. The password field not explicitly disabled with the AUTOCOMPLETE feature.



9:18 PM: Hi Guys,
I got this email from XYZ saying that everyone enrolled in this unit get a free laptop for their studies. Click Register. You guys probably want to hurry because it says there is only 50 machines up for grab.
Enjoy

Figure 6. Moodle message page allows entering malicious links to the message, which allows attackers to conduct phishing attacks. In this case, victims who clicked the link would be redirected to a malicious site like a hypothetical one at malicious-site.com.

usernames against a commonly known password is a way of guessing the username and password of a user account. This method is less likely to cause account lockout. The horizontal brute force attack allows the attackers use a username (e.g. a username that is typically a numeric value 34235123 to 34235150) to attempt to login using a commonly known password. The vertical brute force method is the most commonly known method of brute forcing a user account. This method allows the use single username against list of passwords. If the user has successfully guessed the username and the appropriate password the application may allows access or it might lock the user account by initiating the account lockout function within the application. During the brute for the Moodle, the vertical brute force method was used because the login interface does not employ account lockout as default.

The security risk for this vulnerability is medium.

- Student user: The attacker could be another student that is trying to access a victim student's stored content (assignments, test results, personal information). Successfully hijacked student account could allow the attackers to compromise the confidentiality and the integrity of the protected data that could potentially breach the privacy and damage the reputation of the organization.
- Lecturer user: The attacker could be student that is trying to get unauthorized access to the resources (such as the assignments, results, exam paper) stored within the lecturer's account. If an attacker was able to compromise the lecturer account then they could potentially compromise the confidentiality, integrity of the assignments and exam papers, etc., that could breach the confidentiality, integrity and the reputation of the organization.
- Admin user: The administrative user has access to all the resources within the Moodle framework. The attackers could be any user that is trying to get unauthorized access to all the protected resources within the framework. Successfully hijacked admin account could breach the confidentiality, integrity; availability of the organization and that could potentially disrupt the business function and the reputation of the organization.

4.5. Low risk: weak implementation of CSRF tokens

The Moodle application uses Cross Site Request Forgery (CSRF) tokens in order to prevent attacks against CSRF. However, during the source code analysis it was noted that the CSRF token (*seesskey*) was generated at the start of a session and are remains the same until session termination.

The source code below shows the Moodle source code for generating the session key where the *sesskey* generated at once during session and is 10 character random string from the sessionlib page at `lib/sessionlib.php`:

```
$_SESSION['USER'] -> sesskey
                  = random_string(10);
```

The Cross-site request forgery attack allows the attackers to deceive the victim users to click on a crafted link which in turn allows them to conduct arbitrary action to their user account with the their knowledge. CSRF is an attack that exists because of the application's trust in the browser, which allows the malicious users to replay the same request multiple times. By clicking on the crafted link make uses the existing cookie details stored within the browser to send the request as the victim user without raising an alert.

For instance, the Moodle contains the administrative function where the administrative user has the ability to activate a disabled or newly created account by sending a GET request with valid parameters to the web application. The proper business logic is when the user created a new user account then they have to wait a certain amount to time for the administrative user to activate the user account. Furthermore, if a certain user has been banned from using the Moodle application then their user account has been disabled and the process to activate the account is by getting permission from the administrative user.

The attacker was able to retrieve the sesskey of the user by accessing their cache or my sniffing the network traffic then they could potentially conduct a Cross Site Request Forgery (CSRF) attack on the administrative user to activate a disabled or new user account without following the proper activation process.

Figure 7 shows a proof-of-concept page designed to send the administrative user a crafted link with the appropriate parameters (*unsuspend, sesskey*) to reactivate the suspended user account without the administrative user's knowledge.

This finding is rated low risk because of the following reasons: (1) Application transmits the CSRF token within as an URL parameter; (2) The application is tied to a single session as a result this attack is time sensitive and attacker has a limited window of opportunity to successfully conduct the replay attack using an existing sesskey.

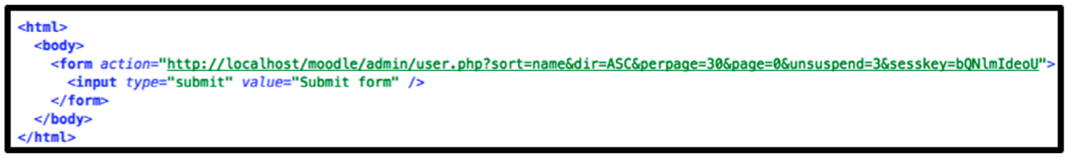
- Student user: The attacker can be another student or a malicious attacker that is trying to enforce the victim user to perform arbitrary actions without their knowledge. For instance, malicious attacker posting content on the forum on behalf of the victim student user.
- Lecturer user: The attacker could be student on a malicious user on the Internet that is trying to get the lecturer to conduct arbitrary actions without their knowledge. For example, Successful exploitation of this vulnerability could allows the attackers to change the assignment result of a certain user within the lecturers' knowledge. Because the POST request that is made to update the assignment results contains the sesskey parameter and attackers could steal this parameter value to replay another request that would potentially allow them change their grades by sending another crafted request with the valid sesskey.
- Admin user: The administrative user has access to all the resources within the Moodle framework. The attackers could be any user that is trying to get unauthorized access to all the protected resources within the framework. Successfully hijacked admin account could breach the confidentiality, integrity, and availability of the organization and that could potentially disrupt the business function and the reputation of the organization. For instance, the attacker could be attempting to get access to the Moodle framework without proper authentication and Moodle contains the self-registration function where the registered user details are submitted and queued for the administration users' permission. By deceiving the administrative user to click on a crafted URL to potentially send a request that enables the malicious user account gives the malicious attacker foothold to the application.

4.6. Low risk: weak segregation of the administrative and normal user functionalities

The administrator and normal users are authenticated through the same page (`/login/index.php`.

Not properly segregating the administrative and normal user (student, teacher, etc.,) could potentially allow malicious users to compromise higher privilege users such as the administrative user.

The administrative user with the Moodle framework has the highest privileges, which allows the user to have access to all protected data (e.g. exam paper, grading, assignments, student records, lecturer records, etc.,). If an administrative user's account is compromised, it would essentially be a malicious attacker's gold mine where the attacker would be in a position to breach the confidentiality, integrity and availability of the Moodle framework and its protected data. That could then disrupt the business operation and cause reputation damages to the organization.



```
<html>
  <body>
    <form action="http://localhost/moodle/admin/user.php?sort=name&dir=ASC&perpage=30&page=0&unsuspend=3&sesskey=bQNlmIdeoU">
      <input type="submit" value="Submit form" />
    </form>
  </body>
</html>
```

Figure 7. The CSRF Proof-of-concept attack – crafted GET request.

4.7. Medium risk: HTTPS enabled only for login page (as default)

Moodle 2.6 is deployed over a clear text protocol (HTTP), which potentially allows malicious users to conduct sniffing-style attacks to retrieve sensitive information from the victim users. Furthermore, the application allows the transport layer security only for the login page, however this option is turned off as default.

As default the application has no secure encryption channel in place, which allows the attackers to conduct sniffing-style attacks to retrieve sensitive information such as usernames, passwords, and session cookies.

This vulnerability is of medium level risk.

- Student user: The attacker could be another student trying to access a victim student's stored content (assignments, test results, and personal information). A successfully hijacked student account could allow the attackers to compromise the confidentiality and the integrity of the protected data that could potentially breach the privacy and damage the reputation of the organization.
- Lecturer user: The attacker could be a student trying to get unauthorized access to the resources (such as the assignments, results, and exam paper) stored within the lecturer's account. If an attacker was able to compromise the lecturer account then they could potentially compromise the confidentiality and integrity of the assignments and exam papers, etc., resulting in the breach of the confidentiality, integrity and the reputation of the organization.
- Admin user: The administrative user has access to all the resources within the Moodle framework. The attacker could be any user trying to get unauthorized access to all the protected resources within the framework. Successfully hijacked admin accounts could breach the confidentiality, integrity, and availability of the organization and that could potentially disrupt the business function and the reputation of the organization.

4.8. Low risk: HTTPOnly flag is not set (as default)

Moodle 2.6 does not employ `HTTPOnly` flag as default. If `HTTPOnly` flag is not set, malicious scripts on the client's browser will be able to access the cookies.

The Moodle stores the users session in two different session tokens: Unauthenticated session token or Authenticated session token. The unauthenticated session token is issued when the Moodle user first access the web application and upon authentication the user is issued with a new session token this security control is implemented by the Moodle in order to avoid session fixation attacks.

Since the users session state is maintained by the session token (MoodleSession) in an event where the attackers were able to steal the session token then they could potentially take over the victim users' user account.

For example, if the Moodle application contains a cross-site scripting vulnerability, the attackers could use it to steal the victim user's session token by injecting java script code that is then executed on the victim users web browser. There are security controls that prevent attacks in this nature, for instance proper input validation prevents malicious users from injecting potentially malicious script code like Javascript and enabling `HTTPOnly` flag. The `HTTPOnly` flag prevents malicious scripts on the client browser to access the cookie values.

This vulnerability is of low level security risk.

- Student user: The attacker could be another student trying to access a victim student's stored content (assignments, test results, and personal information). A successfully hijacked student's account could allow the attackers to compromise the confidentiality and the integrity of the protected data that could potentially breach the privacy and damage the reputation of the organization.
- Lecturer user: The attacker could be a student trying to get unauthorized access to the resources (such as the assignments, results, and exam paper) stored within the lecturer's account. If an attacker was able to compromise the lecturer's account, they could potentially compromise the confidentiality and integrity of the assignments and exam papers, etc., that could then breach the confidentiality, integrity, and the reputation of the organization.
- Admin user: The administrative user has access to all the resources within the Moodle framework. Attackers could be any user trying to get unauthorized access to all the protected resources within the framework. A successfully hijacked administrative account could breach the confidentiality, integrity,

and availability of the organization, which could lead to potentially disrupting the business function and reputation of the organization.

4.9. *Low risk: maximum length of password limited to 15 characters*

The length of user's password is limited to 15 characters. The issue is that the 15-character passwords could be brute forced and the application must have a limitation for the maximum length of the password (CWE-521: Weak Password Requirements) to prevent Denial of Service attacks on the web application server. Short passwords can potentially be easily compromised during a brute force attack. However, it is also important to set a limit to the password length to prevent Denial of Service on the web server. This can be due to the significant resource consumption during the hashing of the password. As a solution to this issue, the maximum length of the password can be set to 32 characters to avoid both brute force attacks and denial of service attack.

This vulnerability is a low level security risk.

- Student user: The attacker could be another student trying to access a victim student's stored content (assignments, test results, and personal information). A successfully hijacked student's account could allow the attackers to compromise the confidentiality and the integrity of the protected data that could potentially breach the privacy and damage the reputation of the organization.
- Lecturer user: The attacker could be a student trying to get unauthorized access to the resources (such as the assignments, results, and exam paper) stored within the lecturer's account. If an attacker was able to compromise the lecturer's account, they could potentially compromise the confidentiality and integrity of the assignments and exam papers, etc., that could then breach the confidentiality, integrity, and the reputation of the organization.
- Admin user: The administrative user has access to all the resources within the Moodle framework. Attackers could be any user trying to get unauthorized access to all the protected resources within the framework. A successfully hijacked administrative account could breach the confidentiality, integrity, and availability of the organization, which could lead to potentially disrupting the business function and reputation of the organization.

5. Conclusions and future work

At the end of the security study of the Moodle framework, we were able to discover three medium risk findings and six low risk findings. It was noted that compromising the three medium risk findings in conjunction with the low risk could potentially allow the malicious user to compromise the confidentiality and integrity of the Moodle framework that could then lead to disruptions to the business operations and breach the privacy of the Moodle users.

Moodle is an out of the box tool developed for e-learning platform, which allows organization to conduct the online learning tasks. A potentially compromised Moodle framework could cause breaches in confidentiality and integrity of the protected data such as exam papers, assignments, and grades that could then breach the privacy and disrupt business functionalities.

By analyzing all the results that were discovered during the security study of Moodle, Moodle 2.6 is not secure as an out of the box product.

Our security-testing framework for testing Moodle or any other PHP-based web application allows businesses to test their Moodle or PHP-based web application security in a comprehensive manner by conducting source code analysis in conjunction with a web application penetration testing.

Many small to medium business who use Moodle as an online teaching platform suffer from lack of funds toward security due to the lack of knowledge in security. This paper shows that it is possible to test big web application like Moodle using free tools without expert knowledge in security or in coding.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Akalanka Karunarathne Mudiyanselage is a cyber security consultant at EY, Australia. He is specialized on penetration testing for security vulnerabilities. Prior to join EY, he completed his Bachelor of IT Security (Honours) at Deakin University. He has a broad interest in creating and improve security software tools.

Lei Pan with Deakin University, Australia. He serves as a course director for cyber security discipline. He has a strong passion in teaching and researching cyber security problems and issues affecting the twenty-first century digital future. He has published more than 50 internationally peer-reviewed journal and conference papers. He is also an active educator on futurelearn.com.

ORCID

Lei Pan http://orcid.org/0000-0002-4691-8330

References

[1] Department of Education and Early Childhood Development. Digital learning support and services; 2017. Available from: http://www.education.vic.gov.au/school/principals/curriculum/pages/elearning.aspx

[2] OWASP Foundation. Top 10 vulnerabilities; 2017. Available from: https://www.owasp.org/index.php/Top_10_2013-Top_10

[3] Stuttard D, Pinto M. The web application Hacker's handbook. 2nd ed. Indianapolis, Indiana, the United States: Wiley; 2011.

[4] Jovanovic N, Kruegel C, Kirda E. Static analysis for detecting taint-style vulnerabilities in web applications. J Comput Secur. 2010;18(5):861–907.

[5] Bacudio AG, Yuan X, Chu BTB, et al. An overview of penetration testing. Int J Network Secur Appl. 2011;3(6):19–38.

[6] Alisherov F, Sattarova F. Methodology for penetration testing. Int J Grid Distrib Comput. 2009;2(2):43–50.

[7] Dahse J. RIPS — a static source code analyser for vulnerabilities in PHP scripts; 2017. Available from: http://php-security.org/downloads/rips.pdf

[8] Dahse J, Holz T. Simulation of built-in PHP features for precise static code analysis. San Diego, CA, the United States: NDSS; 2014.