Chloe Chamberland

March 15, 2021

Several Vulnerabilities Patched in Tutor LMS Plugin

On December 15, 2020, our Threat Intelligence team responsibly disclosed several vulnerabilities in Tutor LMS, a WordPress plugin installed on over 20,000 sites. The first five flaws made it possible for authenticated attackers to inject and execute arbitrary SQL statements on WordPress sites. This made it possible for attackers to obtain information stored in a site's database, including user credentials, site options, and other sensitive information. The remaining flaws made it possible for authenticated attackers to perform several unauthorized actions like escalate user privileges and modify course settings through the use of various AJAX actions.

We initially reached out to Tutor LMS on December 15, 2020. We received a response confirming the appropriate inbox for handling the discussion on December 21, 2020 and supplied the full disclosure details that same day. Tutor LMS acknowledged our report just a day later and released the first set of patches on December 30, 2020. After a few follow ups and a few revised versions, a sufficiently patched version of the plugin was released on February 16, 2021.

Several of the patched vulnerabilities are very severe. Therefore, we highly recommend updating to the patched version, 1.8.3, immediately.

Wordfence Premium users received a firewall rule to protect against any exploits targeting these vulnerabilities on December 15, 2020. Sites still using the free version of Wordfence received the same protection on January 14, 2021. After discovering a way to bypass the existing protection, we released an additional firewall rule on February 25, 2021. Sites still using the free version of Wordfence will receive that protection on March 27, 2021.

Brief Introduction to SQL Injection Vulnerability Types

In today's post, we will take a look at several different types of SQL Injection vulnerabilities that were discovered in Tutor LMS. Before diving into the details of the vulnerabilities, it is important to understand the different types of SQL injection vulnerabilities and how they can be exploited. This section details the differences between the three SQL injection types we will disclose today.

Blind-based SQL injection
A blind SQL injection vulnerability occurs when a SQL statement or query can be added to an already existing SQL query where the response will only provide a true or false answer rather than providing the full results of a query. An attacker can use this to pull information from a database by pulling one character at a time using specially crafted substring function queries.

Time-based SQL Injection
A time-based SQL Injection vulnerability occurs when a SQL statement or query can be added to an already existing SQL query, however, no data can be gathered explicitly from a requests response. Instead, you must rely on the use of time-based SQL functions like `SLEEP()` and `WAITFOR()` while observing the response time to obtain results of the query from the database. Just like with blind-based SQL Injection, an attacker would use this to pull information from a database one character at a time using specially crafted queries containing time-based functions.

UNION-based SQL Injections
A UNION-based SQL Injection vulnerability occurs when an additional SQL query can be added to an already existing SQL query as a UNION. This differs from the previous two SQL injection types discussed because data can easily be extracted by simply adding an additional query to the already existing query through the use of the UNION operator. This is one of the simplest, and easiest, forms of SQL Injection vulnerability that can be exploited. An attacker could use this type of SQLi to pull data from anywhere in the database using a simple query like `SELECT * FROM wp_users;`. With that query, all rows from the `wp_users` table would be returned.

It should be noted that a UNION query will need to retrieve the same amount of rows from the database as the original query, therefore, in most cases exploiting this type of SQLi vulnerability can require some trial and error.

If you would like to learn more about how SQL Injection attacks work, please visit our learning center here.

The SQL Injection Vulnerabilities

Description: SQL Injection [Blind/Time based]
Affected Plugin: Tutor LMS
Plugin Slug: tutor
Affected Versions: <= 1.7.6
CVE ID: CVE-2021-24185
CVSS Score: 6.5 (Medium)
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Fully Patched Version: 1.7.7

Tutor LMS is a robust learning management system designed to simplify selling and creating online courses with WordPress. There are many useful features like the ability to create and customize courses with different testing options, easy user and teacher registration, the ability to leave reviews for courses, and much more.

Tutor LMS allows students to leave reviews for courses. A user does need to be authenticated in order to leave a review, however, it is very easy to register as a student on sites running the Tutor LMS plugin.

In order to enter ratings on courses, the plugin registers an AJAX action, `wp_ajax_tutor_place_rating`, tied to the `tutor_place_rating` function. This function will process the request and, if a review already exists for the current user and course, it will update the rating. However, if a review does not exist, it will create a new review and update the database with that review.

```
public function tutor_place_rating(){
    global $wpdb;
```

```
    $course_id = sanitize_text_field(tutor_utils()->avalue_dot('course_id', $_POST));
    $review = wp_kses_post(tutor_utils()->avalue_dot('review', $_POST));

    $user_id = get_current_user_id();
    $user = get_userdata($user_id);
    $date = date("Y-m-d H:i:s", tutor_time());

    do_action('tutor_before_rating_placed');

    $previous_rating_id = $wpdb->get_var("select comment_ID from {$wpdb->comments} WHERE comment_post_ID={$course_id}

    $review_ID = $previous_rating_id;
    if ( $previous_rating_id){
        $wpdb->update( $wpdb->comments, array('comment_content' => $review),
            array('comment_ID' => $previous_rating_id)
        );

        $rating_info = $wpdb->get_row("SELECT * FROM {$wpdb->commentmeta} WHERE comment_id = {$previous_rating_id} AN
        if ($rating_info){
            $wpdb->update( $wpdb->commentmeta, array('meta_value' => $rating), array('comment_id' => $previous_rating
        }else{
            $wpdb->insert( $wpdb->commentmeta, array('comment_id' => $previous_rating_id, 'meta_key' => 'tutor_rating
        }
```

```
    $data = array('msg' => __('Rating placed success', 'tutor'), 'review_id' => $review_ID, 'review' => $review);
    wp_send_json_success($data);
}
```

By using `get_var()` without the use of `prepare()` when checking for the existence of a review, along with no SQL sanitization on the user supplied variables, a user could inject arbitrary SQL statements while leaving a review.

This vulnerability would need to be exploited using blind-based SQLi techniques. More specifically, the user supplied data for the `course_id` parameter could include a boolean statement, which would be something like `1=1`, which is always true and `1=2`, which is always false. If the response indicated that a review had been updated, then the answer to the SQL query would be true. If the response indicated that a new review had been created then the answer to the SQL query would be false.

These arbitrary SQL statements could allow an attacker to retrieve information from the database such as usernames, passwords or other sensitive information. In some cases, where a MySQL server is insecurely configured, this could allow an attacker to read files and create new files containing webshells along with modifying information in the database.

The vulnerability could also be exploited using a time-based method.

Description: SQL Injection [Time-Based]
Affected Plugin: Tutor LMS
Plugin Slug: tutor
Affected Versions: <= 1.7.6
CVE ID: CVE-2021-24181
CVSS Score: 6.5 (Medium)
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Fully Patched Version: 1.7.7

Another feature of Tutor LMS is the ability for teachers to mark answers as correct once they have been submitted by a student. In order to provide this functionality, the plugin registered an AJAX action, `wp_ajax_tutor_mark_answer_as_correct`, tied to the `tutor_mark_answer_as_correct` function.

```
public function tutor_mark_answer_as_correct(){
    global $wpdb;

    $answer_id = sanitize_text_field($_POST['answer_id']);
    $inputValue = sanitize_text_field($_POST['inputValue']);

    $answer = $wpdb->get_row("SELECT * FROM {$wpdb->prefix}tutor_quiz_question_answers WHERE answer_id = {$answer_i
    if ($answer->belongs_question_type === 'single_choice'){
        $wpdb->update($wpdb->prefix.'tutor_quiz_question_answers', array('is_correct' => 0), array('belongs_questio
    }
    $wpdb->update($wpdb->prefix.'tutor_quiz_question_answers', array('is_correct' => $inputValue), array('answer_id
}
```

Before updating the database to mark an answer as correct, the function used `get_row()` to retrieve the initial answer recorded in the database while using the user-supplied value from the POST parameter `answer_id` as the answer ID. Unfortunately, there was no SQL sanitization on the user supplied value, nor was the function using a prepared statement, making it possible for SQL queries to be injected.

Due to the fact that this query happens prior to two other queries, an immediate response could not be observed and effectively required the use of time-based SQL queries in order to retrieve any data from the database.

This functionality was intended to be used by teachers and administrators only, however, since it was an AJAX action with no nonce protection or capability checks in place, this meant that any authenticated user, including students, had the ability to execute this action and exploit the SQL injection vulnerability.

Description: SQL Injection [UNION]
Affected Plugin: Tutor LMS
Plugin Slug: tutor
Affected Versions: <= 1.8.2
CVE ID: CVE-2021-24182, CVE-2021-24183, CVE-2021-24186
CVSS Score: 6.5 (Medium)
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Fully Patched Version: 1.8.3

In addition to the two previous vulnerabilities that we discovered, we also found three UNION-based SQL injection vulnerabilities.

Union-based SQL Injection #1

Tutor LMS allows teachers to retrieve a set of answers for a given question while analyzing the response of students. In order to provide this functionality, the plugin registers an AJAX action `wp_ajax_tutor_quiz_builder_get_answers_by_question` tied to the `tutor_quiz_builder_get_answers_by_question` function.

```
public function tutor_quiz_builder_get_answers_by_question(){
global $wpdb;
$question_id = sanitize_text_field($_POST['question_id']);
$question_type = sanitize_text_field($_POST['question_type']);

$question = $wpdb->get_row("SELECT * FROM {$wpdb->prefix}tutor_quiz_questions WHERE question_id = {$question_id} ");
$answers = $wpdb->get_results("SELECT * FROM {$wpdb->prefix}tutor_quiz_question_answers where belongs_question_id = {

ob_start();

switch ($question_type){
    case 'true_false':
        echo '<label>'.__('Answer options &amp; mark correct', 'tutor').'</label>';
        break;
    case 'ordering':
        echo '<label>'.__('Make sure you're saving the answers in the right order. Students will have to match this o
        break;
}
```

The function used `get_results()` to obtain the answers from the database. Again, there was no SQL sanitization on the user supplied input, nor was there any use of prepared statements. This made it possible for an attacker to supply a UNION query in the `question_id` parameter that would execute and provide the direct results of the query in the response to the request.

This function could also be exploited using blind and time-based SQLi techniques.

Union-based SQL Injection #2

Another feature of Tutor LMS is the ability to build quizzes as a teacher on a site. In order to provide this functionality, the plugin registered various AJAX actions to make the quiz building process easy and require fewer page reloads. One `tutor_quiz_builder_get_question_form` function.

This AJAX action was designed to retrieve the already existing questions for a quiz while editing them in the quiz builder.

```
public function tutor_quiz_builder_get_question_form(){
    global $wpdb;
    $quiz_id = sanitize_text_field($_POST['quiz_id']);
    $question_id = sanitize_text_field(tutor_utils()->avalue_dot('question_id', $_POST));

    if ( ! $question_id){
        $next_question_id = tutor_utils()->quiz_next_question_id();
        $next_question_order = tutor_utils()->quiz_next_question_order_id($quiz_id);

        $new_question_data = array(
            'quiz_id'               => $quiz_id,
            'question_title'        => __('Question', 'tutor').' '.$next_question_id,
            'question_description'  => '',
            'question_type'         => 'true_false',
            'question_mark'         => 1,
            'question_settings'     => maybe_serialize(array()),
            'question_order'        => $next_question_order,
        );

        $wpdb->insert($wpdb->prefix.'tutor_quiz_questions', $new_question_data);
        $question_id = $wpdb->insert_id;
    }

    $question = $wpdb->get_row("SELECT * FROM {$wpdb->prefix}tutor_quiz_questions where question_id = {$question_id}

    ob_start();
    include  tutor()->path.'views/modal/question_form.php';
    $output = ob_get_clean();

    wp_send_json_success(array('output' => $output));
}
```

When the `question_id` parameter is supplied, the function uses `get_row()` to obtain the answer data from the database. There was no SQL sanitization on the user supplied input, nor did it use a prepared statement. This made it possible for an attacker to supply a UNION query in the `question_id` parameter that would execute and provide the results of the query in the response of the request.

This function, along with the `tutor_quiz_builder_get_answers_by_question()` function, were intended to be for instructor and administrator use only. Unfortunately, however, since they were AJAX actions with no nonce protection or capability checks in place, any authenticated user, including students, had the ability to execute this action and exploit the SQL injection vulnerability.

Union-based SQL Injection #3

The last SQL injection vulnerability we discovered was more unique. As previously mentioned, Tutor LMS allowed course instructors to create quizzes for students to take. Whenever a student takes a quiz, the plugin records the results of that attempt in the database, and also makes it possible for students to later revisit those results. In order to provide this functionality, the plugin created the function `get_answer_by_id` that would retrieve the results of a quiz when a user accessed the quiz attempt details.

```
public function get_answer_by_id($answer_id){
        global $wpdb;

        if (is_array($answer_id)){
            $in_ids = implode(",", $answer_id);
            $sql = "answer.answer_id IN({$in_ids})";
        }else{
            $sql = "answer.answer_id = {$answer_id}";
        }

        $answer = $wpdb->get_results("SELECT answer.*, question.question_title, question.question_type
        FROM {$wpdb->prefix}tutor_quiz_question_answers answer
        LEFT JOIN {$wpdb->prefix}tutor_quiz_questions question ON answer.belongs_question_id = question.question_id
        WHERE 1=1 AND {$sql} ");

        return $answer;
```

While retrieving those results, the function used `get_results()` to retrieve the results from the database. Due to the fact that there was no SQL escaping on the quiz answers as they were recorded, SQL statements could be included as a quiz response. Once the data was retrieved from the database upon accessing the attempt details page, the stored SQL statements would execute and supply the requested information from the database.

Unprotected AJAX Endpoints

In addition to several SQL Injection vulnerabilities, we discovered a number of unprotected AJAX endpoints that could allow low-level users like students to perform a plethora of actions that allowed them to create new quizzes, modify course information, change grades, escalate privileges and more. The following privilege escalation is the most significant endpoint that we discovered.

Description: Unprotected AJAX Action to Privilege Escalation
Affected Plugin: Tutor LMS
Plugin Slug: tutor
Affected Versions: <= 1.7.6
CVE ID: CVE-2021-24184
CVSS Score: 8.1 (High)
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N
Fully Patched Version: 1.7.7

Tutor LMS creates two new roles upon activation of the plugin: the "student" role and the "instructor" role. The student role is designed for users who want to register for courses, while the instructor role is designed for users who want to be instructors and host course content on a site.

Two features the plugin offers are the ability to allow students to request to become a teacher and the ability for administrators to directly create new instructors on a given site. Unfortunately, both of these features were insecurely implemented.

In order to request to become a teacher, a user would need to submit a request that sends a notification to the site administrator for approval. Once approved, the user would then be granted the instructor privileges and be able to perform all instructor actions. Unfortunately, the approval process was vulnerable due to a lack of a capability check and authenticated students could approve themselves as instructors, bypassing this approval process.

Additionally, administrators have the option to add new instructors outside of the standard WordPress new user functionality. Unfortunately, there was no capability check on this AJAX action so any authenticated user could add a new instructor account and then use that to create potentially malicious content on a site.

Both of these vulnerabilities did require that a user be able to obtain a nonce from the `/wp-admin` dashboard.

Disclosure Timeline

December 15, 2020 – Conclusion of the plugin analysis that led to the discovery of several vulnerabilities in Tutor LMS. We develop firewall rules to protect Wordfence customers and release them to Wordfence Premium users. We initiate contact with Tutor LMS.
December 21, 2020 – We receive a response confirming the appropriate inbox for handling discussion.
December 21, 2020 – We provide full disclosure details.
December 22, 2020 – Tutor LMS confirms receipt of our disclosure and begins working on fix.
December 30, 2020 – An initial patch is released.
January 4, 2021 – We analyze the initial patch and determine that further patching is required.
January 14, 2021 – Free Wordfence users receive firewall rule.
January 15, 2021 – We follow-up to check on the status of the patches.
January 15, 2021 to February 16, 2021 – We continue discussion to ensure all vulnerabilities have been patched and measures to harden the security of the plugin are made. There are several patches released during this time.

February 16, 2021 – A final and fully patched version of Tutor LMS is released as version 1.8.3.
February 25, 2021 – We discover a bypass to the original SQL injection protection and develop a new firewall rule to
March 27, 2021 – Free Wordfence users receive additional firewall rule.

Conclusion

In today's post, we detailed several flaws in Tutor LMS that granted attackers the ability to retrieve or modify arbitrary data in the database and elevate privileges. These flaws have been fully patched in version 1.8.3. We recommend that users immediately update to the latest version available, which is version 1.8.5 at the time of this publication.

Wordfence Premium users received firewall rules protecting against these vulnerabilities on December 15, 2020 and February 25, 2021, while those still using the free version of Wordfence received the initial protection on January 14, 2020 and will receive the additional protection on March 27, 2021.

If you know a friend or colleague who is using this plugin on their site, we highly recommend forwarding this advisory to them to help keep their sites protected as there were several severe vulnerabilities patched in this plugin.

Special thanks to the team at Tutor LMS for working diligently to secure their plugin and quickly responding to any comments for further security requests we had.

Did you enjoy this post? Share it!

Comments

1 Comment

Kawshar Ahmed *
March 15, 2021
12:19 pm

On behalf of the Themeum team, I think you very much for your amazing help.

Breaking WordPress Security Research in your inbox as it happens.

you@example.com

By checking this box I agree to the terms of service and privacy policy.*

SIGN UP

Our business hours are 9am-8pm ET, 6am-5pm PT and 2pm-1am UTC/GMT excluding weekends and holidays.
Response customers receive 24-hour support, 365 days a year, with a 1-hour response time.

Terms of Service Privacy Policy
CCPA Privacy Notice

Products
Wordfence Free
Wordfence Premium
Wordfence Care
Wordfence Response
Wordfence Central

Support
Documentation
Learning Center
Free Support
Premium Support

News
Blog
In The News
Vulnerability Advisories

About
About Wordfence
Careers
Contact
Security
CVE Request Form

Stay Updated

Sign up for news and updates from our panel of experienced security professionals.

you@example.com

By checking this box I agree to the terms of service and privacy policy.*

SIGN UP

© 2012-2022 Defiant Inc. All Rights Reserved