Danupon Thepsing

[ACS EDU - Evaluation Report] Analysis of CVE-2022-3141

This evaluation report has been prepared by Danupon Thepsing as part of the ACS EDU course, focusing on the topic of Real-World Web Exploitation. The report provides an in-depth analysis of CVE-2022-3141.

Published: 2022-09-19
Updated: 2023-03-27
Title: Translatepress Multilinugal < 2.3.3 - Admin+ SQLi

Description

The Translate Multilingual sites WordPress plugin before 2.3.3 is vulnerable to an authenticated SQL injection. By adding a new language (via the settings page) containing specific special characters, the backticks in the SQL query can be surpassed and a time-based blind payload can be injected.

CWE

CWE-89 SQL Injection

CVSS

Score	Severity	Version	Vector String
8.8	**HIGH**	3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

VENDORS

Unknown

PRODUCTS

Translate Multilingual sites – TranslatePress
affected from **2.3.3** before **2.3.3**

Reporter

Elias Hohl

Impact

Data Breach: Unauthorized access to sensitive data, leading to information theft, financial loss, and legal implications.

Website Defacement: Altering the appearance of a website, which can damage an organization's reputation and trustworthiness.

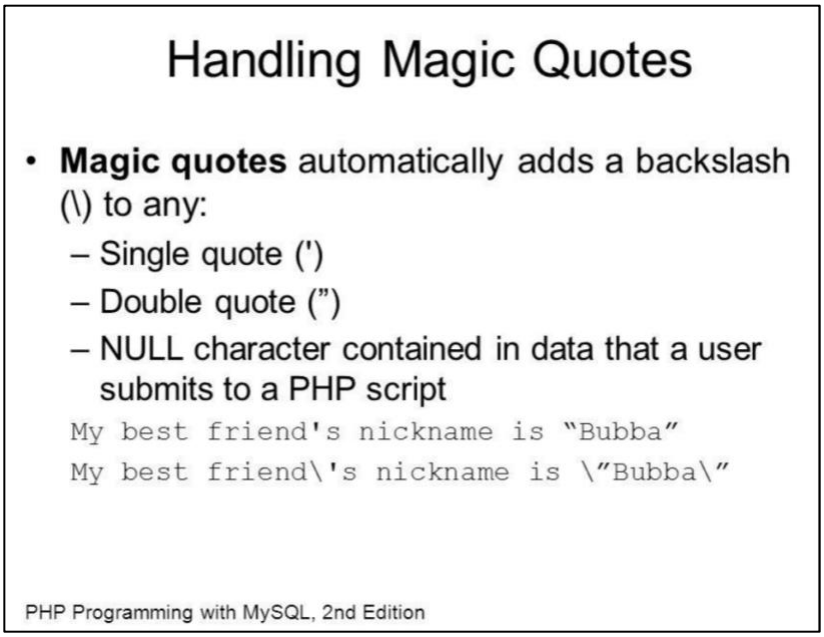
Malware Distribution: Using compromised websites to distribute malware, affecting visitors and potentially leading to further compromises.

SEO Spam: Injecting malicious content or links to manipulate search engine rankings, damaging the site's credibility and visibility.

8.8 CVSS score suggests that an attacker could exploit this vulnerability with low complexity, requiring minimal privileges and no user interaction. The impacts are broad, potentially compromising the confidentiality, integrity, and availability of the system to a high degree. This underscores the urgency for administrators and developers to apply patches or mitigative measures immediately.

Proof of Concept (PoC)

a. Vulnerability Analysis



WordPress's "magic quotes" feature applies the `addslashes` PHP function to all GET and POST input, escaping quotes and backslashes. This measure is designed to prevent attackers from breaking out of quotes in SQL queries, thereby thwarting many classic SQL injection attacks.

However, this mechanism is not foolproof, especially in scenarios where user input is expected to be numeric or when the application logic fails to properly sanitize input.

How is it vulnerable?

1. Input Handling in TranslatePress

• The plugin retrieves the language code from user input, which is then passed through a function called `get_table_name`.

• Although inputs undergo sanitization via `sanitize_text_field`, this function does not escape backticks, which are crucial to the exploit.

2. Exploiting the Vulnerability

• In the vulnerable code, the table name derived from the user input is encapsulated only in backticks in the SQL query.

• An attacker can manipulate the language input field to inject SQL payloads. For example, using x=%23 (where %23 is a URL-encoded comment character #) allows truncation of the remaining query, potentially leading to data leakage or modification.

Code Review

```
3   /**
4    * Class TRP_Query
5    *
6    * Queries for translations in custom trp tables.
7    *
8    */
9   class TRP_Query{
10
11      protected $table_name;
12      public $db;
13      protected $settings;
14      protected $translation_render;
15      protected $error_manager;
16      protected $tables_exist = array();
17
18      const NOT_TRANSLATED = 0;
19      const MACHINE_TRANSLATED = 1;
20      const HUMAN_REVIEWED = 2;
21
22      const BLOCK_TYPE_REGULAR_STRING = 0;
23      const BLOCK_TYPE_ACTIVE = 1;
24      const BLOCK_TYPE_DEPRECATED = 2;
25
```

The `TRP_Query` class is designed for managing translation data within a WordPress plugin context, providing mechanisms for retrieving, checking, and storing translations in a structured and secure manner. It emphasizes the importance of sanitizing inputs and preparing SQL queries to prevent injection attacks, following best practices for WordPress plugin development.

```php

        /**
         * Return an array of all the active translation blocks
         *
         * @param $language_code
         *
         * @return array|null|object
         */
    public function get_all_translation_blocks( $language_code ){
        $query = "SELECT original, id, block_type, status FROM `" . sanitize_text_field( $this->get_table_name( $language_code ) ) . "` WHE
            $dictionary = $this->db->get_results( $query, OBJECT_K );
            return $dictionary;
    }

```

Retrieves all active and deprecated translation blocks for a specified language from the database. It constructs an SQL query to select relevant data based on the `block_type` and executes it, returning the results as an associative array of objects.

```php
class TRP_Query{

    /**
     * Returns the translations for the provided strings.
     *
     * Only returns results where there actually is a translation ( != NOT_TRANSLATED )
     *
     * @param array $strings_array      Array of original strings to search for.
     * @param string $language_code     Language code to query for.
     * @return object                   Associative Array of objects with translations where key is original string.
     */
    public function get_existing_translations( $strings_array, $language_code, $block_type = null ){
        if ( !is_array( $strings_array ) || count ( $strings_array ) == 0 ){
            return array();
        }
        if ( $block_type == null ){
                $and_block_type = "";
        }else {
                $and_block_type = " AND block_type = " . $block_type;
        }
        $query = "SELECT original,translated, status FROM `" . sanitize_text_field( $this->get_table_name( $language_code ) ) . "` WHERE st

        $placeholders = array();
        $values = array();
        foreach( $strings_array as $string ){
            $placeholders[] = '%s';
            $values[] = $string;
        }

        $query .= "( " . implode ( ", ", $placeholders ) . " )";
            $prepared_query = $this->db->prepare( $query, $values );
        $dictionary = $this->db->get_results( $prepared_query, OBJECT_K  );

        $this->maybe_record_automatic_translation_error(array( 'details' => 'Error running get_existing_translations()' ) );
        if ( is_array( $dictionary ) && count( $dictionary ) === 0 && !$this->table_exists($this->get_table_name( $language_code )) ){
            // if table is missing then last_error is empty for the select query
            $this->maybe_record_automatic_translation_error(array( 'details' => 'Missing table ' . $this->get_table_name( $language_code ) 
        }
        if ($this->db->last_error !== '')
            $dictionary = false;

        return apply_filters( 'trp_get_existing_translations', $dictionary, $prepared_query, $strings_array );
    }
```

Looks for translations of the provided strings in a specified language, optionally filtered by block type. It dynamically creates an SQL query to select only those strings that have translations (i.e., status is not `NOT_TRANSLATED`). This method handles SQL injection prevention by using prepared statements for the inputs.

```php
        /**
     * Check if table for specific language exists.
     *
     * If the table does not exists it is created.
     *
     * @param string $language_code
     */
    public function check_table( $default_language, $language_code ){
        $table_name = sanitize_text_field( $this->get_table_name( $language_code, $default_language ) );
        if ( $this->db->get_var( "SHOW TABLES LIKE '$table_name'" ) != $table_name ) {
            // table not in database. Create new table
            $charset_collate = $this->db->get_charset_collate();

            $sql = "CREATE TABLE `" . $table_name . "`(
                                id bigint(20) AUTO_INCREMENT NOT NULL PRIMARY KEY,
                                original  longtext NOT NULL,
                                translated  longtext,
                                status int(20) DEFAULT " . $this::NOT_TRANSLATED .",
                                block_type int(20) DEFAULT " . $this::BLOCK_TYPE_REGULAR_STRING .",
                                UNIQUE KEY id (id) )
                                 $charset_collate;";
            require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
            dbDelta( $sql );

            $sql_index = "CREATE INDEX index_name ON `" . $table_name . "` (original(100));";
            $this->db->query( $sql_index );

            $this->maybe_record_automatic_translation_error(array( 'details' => 'Error creating regular tables' ) );
            if ( $this->db->get_var( "SHOW TABLES LIKE '$table_name'" ) != $table_name ) {
                // table still doesn't exist after creation
                $this->maybe_record_automatic_translation_error(array( 'details' => 'Error creating regular strings tables' ), true );
            }else {
                // full text index for original
                $sql_index = "CREATE FULLTEXT INDEX original_fulltext ON `" . $table_name . "`(original);";
                $this->db->query( $sql_index );

                //syncronize all translation blocks.
                $this->copy_all_translation_blocks_into_table($default_language, $language_code);
            }
        }else{
                $this->check_for_block_type_column( $language_code, $default_language );
        }
    }
```

Ensures that a specific language table exists in the database; if not, it creates it. This method first checks for the existence of a table for the given language code and, if missing, proceeds to create it with the appropriate schema. It includes steps to handle errors and record them if the table creation fails. Additionally, it manages indexing for efficient data retrieval and ensures that the translation blocks are properly synchronized.

```php
    /**
     * Check if table for machine translation logs exists.
     *
     * If the table does not exists it is created.
     *
     * @param string $language_code
     */
    public function check_machine_translation_log_table(){
        $table_name = $this->db->prefix . 'trp_machine_translation_log';
        if ( $this->db->get_var( "SHOW TABLES LIKE '$table_name'" ) != $table_name )
        {
            // table not in database. Create new table
            $charset_collate = $this->db->get_charset_collate();

            $sql = "CREATE TABLE `{$table_name}`(
                                id bigint(20) AUTO_INCREMENT NOT NULL PRIMARY KEY,
                                url text,
                                timestamp datetime DEFAULT '0000-00-00 00:00:00',
                                strings longtext,
                                characters text,
                                response longtext,
                                lang_source text,
                                lang_target text,
                                UNIQUE KEY id (id) )
                                 {$charset_collate};";
            require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
            dbDelta( $sql );
            $this->maybe_record_automatic_translation_error(array( 'details' => 'Error creating machine translation log tables' ) );

            if ( $this->db->get_var( "SHOW TABLES LIKE '$table_name'" ) != $table_name )
            {
                $this->maybe_record_automatic_translation_error(array( 'details' => 'Error creating machine translation log tables' ), true
                // something failed. Table still doesn't exist.
                return false;
            }
            // table exists
            return true;
        }
        //table exists
        return true;
    }
```

This function checks if a table for logging machine translation activities exists in the WordPress database. If the table does not exist, it creates a new table with the specified schema. The table includes fields for storing information about each translation request, such as the URL, timestamp, strings involved, the response, and source and target languages. This table is crucial for tracking and analyzing how machine translations are performed and can aid in debugging or optimizing the translation process. It uses WordPress's database layer for operations and `dbDelta for creating the table, ensuring compatibility and preventing SQL injection.

```php
234     public function copy_all_translation_blocks_into_table( $default_language, $language_code ){
235         $all_table_names = $this->get_all_table_names( $default_language, array( $language_code ) );
236         if ( count( $all_table_names ) > 0 ){
237                 $source_table_name = $all_table_names[0];
238
239                 // copy translation blocks from table name of this language
240                 $source_language = apply_filters( 'trp_source_language_translation_blocks', '', $default_language, $language_code );
241                 if ( $source_language != '' ){
242                         $source_table_name = $this->get_table_name( $source_language, $default_language );
243                 }
244
245                 $destination_table_name = $this->get_table_name( $language_code, $default_language );
246
247                 // get all tb from $source_table_name and copy to $destination_table_name
248                 $sql = 'INSERT INTO `' . $destination_table_name . '` SELECT NULL, original, "", ' . $this::NOT_TRANSLATED . ', block_ty
249                 $this->db->query( $sql );
250             }
251     }
```

This method is designed to duplicate translation blocks from one language to another within the database. This can be particularly useful when initializing translations for a new language by using existing translations as a base.

```php
260     public function check_gettext_table( $language_code ){
261         $table_name = sanitize_text_field( $this->get_gettext_table_name($language_code) );
262         if ( $this->db->get_var( "SHOW TABLES LIKE '$table_name'" ) != $table_name ) {
263             // table not in database. Create new table
264             $charset_collate = $this->db->get_charset_collate();
265
266             $sql = "CREATE TABLE `" . $table_name . "`(
267                                 id bigint(20) AUTO_INCREMENT NOT NULL PRIMARY KEY,
268                                 original  longtext NOT NULL,
269                                 translated  longtext,
270                                 domain  longtext,
271                                 status int(20),
272                                 UNIQUE KEY id (id) )
273                                  $charset_collate;";
274             require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
275             dbDelta( $sql );
276
277             $this->maybe_record_automatic_translation_error(array( 'details' => 'Error creating gettext strings tables' ) );
278
279             $sql_index = "CREATE INDEX index_name ON `" . $table_name . "` (original(100));";
280             $this->db->query( $sql_index );
281
282             // full text index for original
283             $sql_index = "CREATE FULLTEXT INDEX original_fulltext ON `" . $table_name . "`(original);";
284             $this->db->query( $sql_index );
285         }
286     }
```

Similar to the other check - and -create functions, this method verifies the existence of a gettext table for a specific language. Gettext is a widely used internationalization and localization system that allows for the easy translation of programs. If the table does not exist, it creates one with the appropriate schema.

```php
724     public function get_all_table_names ( $original_language, $exception_translation_languages = array() ){
725             foreach ( $exception_translation_languages as $key => $language ){
726                     $exception_translation_languages[$key] = $this->get_table_name( $language, $original_language );
727             }
728             $return_tables = array();
729             $table_name = $this->get_table_name( ''  );
730             $table_names = $this->db->get_results( "SHOW TABLES LIKE '$table_name%'", ARRAY_N );
731             foreach ( $table_names as $table_name ){
732                 if ( isset( $table_name[0]) && ! in_array( $table_name[0], $exception_translation_languages ) ) {
733                             $return_tables[] = $table_name[0];
734                     }
735             }
736             return $return_tables;
737     }
738
739     public function get_all_gettext_table_names(){
740         global $wpdb;
741         $table_name = $wpdb->get_blog_prefix() . 'trp_gettext_';
742         $return_tables = array();
743
744         $table_names = $this->db->get_results( "SHOW TABLES LIKE '$table_name%'", ARRAY_N );
745         foreach ( $table_names as $table_name ){
746             if ( isset( $table_name[0]) ) {
747                 $return_tables[] = $table_name[0];
748             }
749         }
750         return $return_tables;
751     }
```

Retrieves the names of all tables associated with translation strings for a given original language, excluding any specified exceptions. This method supports functionality like table management or batch operations across multiple language tables.

```php
753         public function update_translation_blocks_by_original( $table_names, $original_array, $block_type ) {
754
755                 $values = array();
756                 foreach( $table_names as $table_name ){
757                         $placeholders = array();
758                         foreach( $original_array as $string ){
759                                 $placeholders[] = '%s';
760                                 $values[] = trp_full_trim( $string );
761                         }
762                 }
763
764                 $placeholders = "( " . implode ( ", ", $placeholders ) . " )";
765                 $query = 'UPDATE `' . implode( $table_names, '`, `' ) . '` SET `' . implode( $table_names, '`.block_type=' . $block_type .
766
767                 return $this->db->query( $this->db->prepare( $query, $values ) );
768         }
```

Updates translation block types for given original strings across specified tables. This method can adjust the classification or status of translation entries in bulk, based on their original text content.

```
780         public function remove_duplicate_rows_in_dictionary_table( $language_code, $batch ){
781                 $table_name = $this->get_table_name( $language_code );
782                 $query = '        DELETE `b`
783                                         FROM
784                                             ' . $table_name . ' AS `a`,
785                                             ' . $table_name . ' AS `b`
786                                         WHERE
787                                             -- IMPORTANT: Ensures one version remains
788                                             `a`.ID < ' . $batch . '
789                                             AND `b`.ID < ' . $batch . '
790                                             AND `a`.`ID` < `b`.`ID`
791
792                                             -- Check for all duplicates. Binary ensure case sensitive comparison
793                                             AND (`a`.`original` = BINARY `b`.`original` OR `a`.`original` IS NULL AND `b`.`original` IS NULL
794                                             AND (`a`.`translated` = BINARY`b`.`translated` OR `a`.`translated` IS NULL AND `b`.`translated`
795                                             AND (`a`.`status` = `b`.`status` OR `a`.`status` IS NULL AND `b`.`status` IS NULL)
796                                             AND (`a`.`block_type` = `b`.`block_type` OR `a`.`block_type` IS NULL AND `b`.`block_type` IS NUL
797                                             ;';
798                 return $this->db->query( $query );
799         }
```

A utility function for cleaning up the translation tables by removing duplicate entries, ensuring each unique string or translation pair is stored only once. This helps maintain the database's integrity and optimizes performance.

Exploit Steps

Proof of Concept

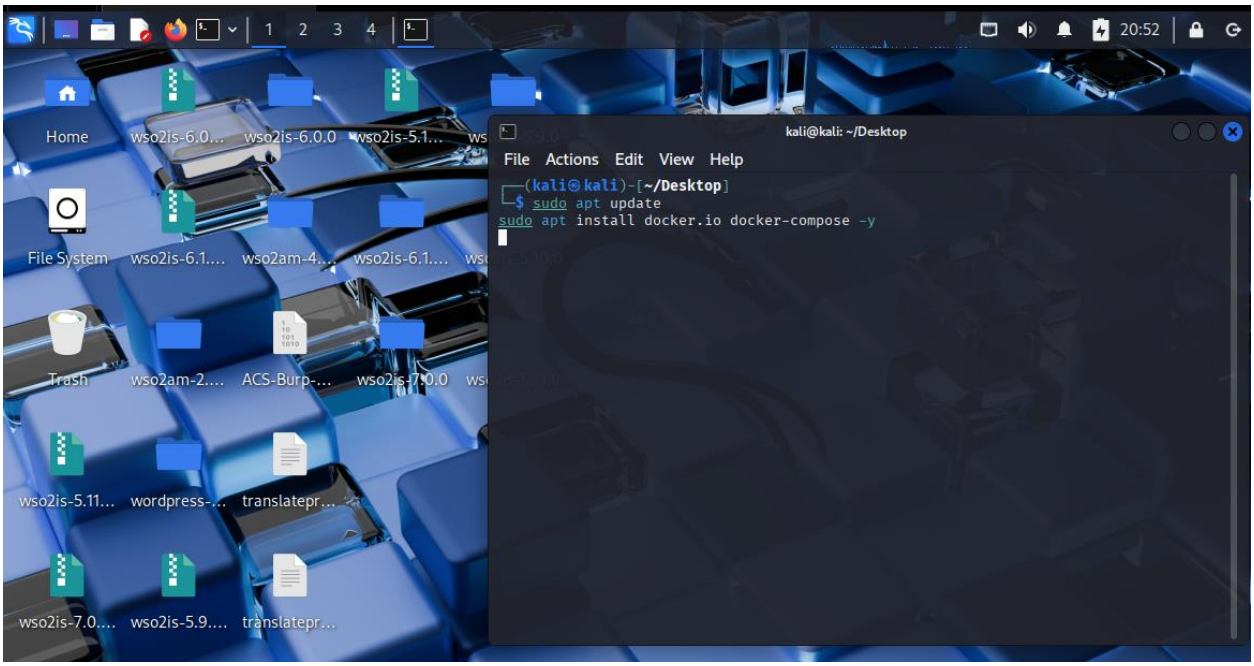
To exploit the vulnerability, someone must send a specifically crafted request adding a new language containing specific special characters, and then open another page and measure the response time to retrieve data. This can be automated via sqlmap.

Environment Setup

The vulnerability was reproduced in the following controlled test environment:

- Attacker Machine**:**
  - OS: Kali Linux
  - Platform: Virtual Machine (VMware)
  - Tools used:Burp Suite, browser, sqlmap, docker
- Target Server**:**
  - “Translatepress Multilingual” Wordpress plugin ver. < 2.3.3
  - Host: Deployed locally or within internal test network

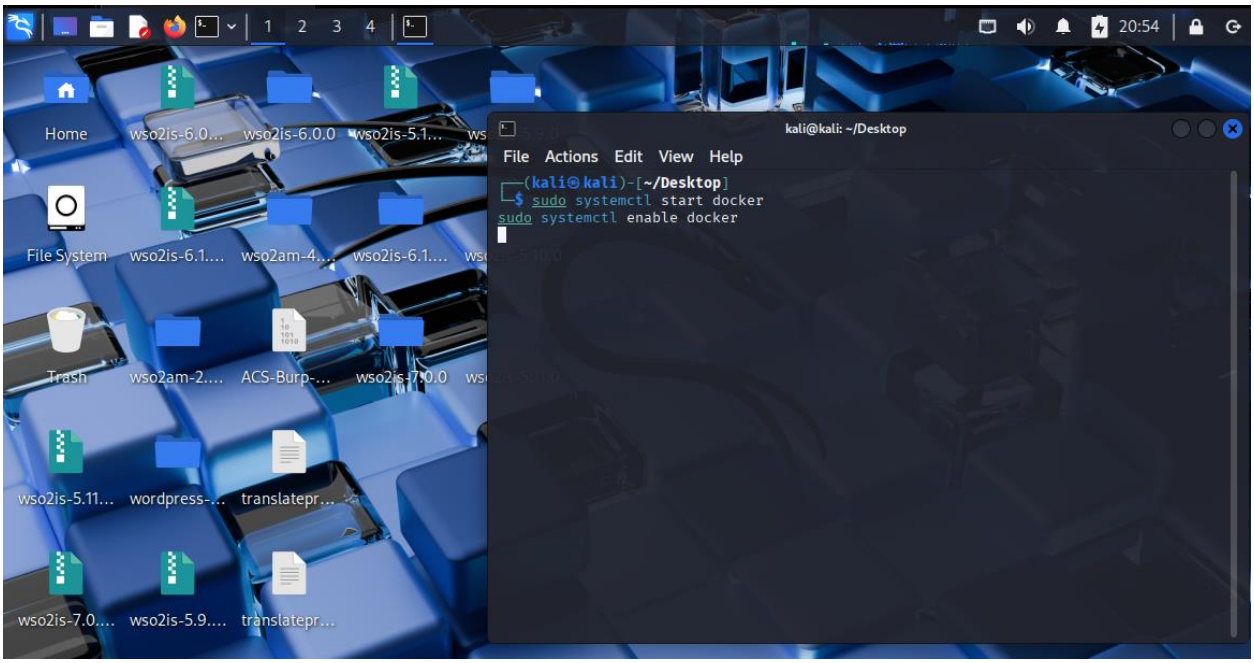
1. Install Docker



Open the Command Line Interface
Begin by launching the terminal on your Kali Linux system.

Update the System Package Index
Run the following command to ensure your package list is up to date:
sudo apt update

Install Docker and Docker Compose
Use the following command to install Docker and Docker Compose:
sudo apt install docker.io docker-compose -y

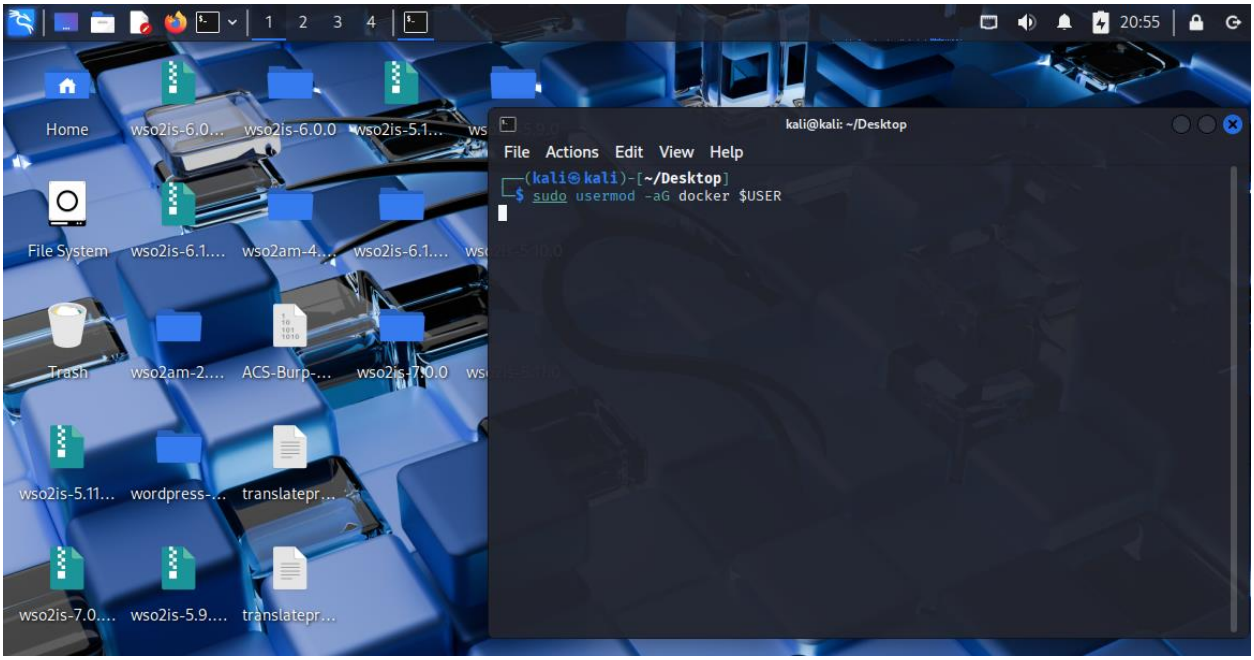


Start and Enable Docker Service

Once Docker is installed, start the Docker service and enable it to launch at boot using the following commands:

sudo systemctl start docker

sudo systemctl enable docker

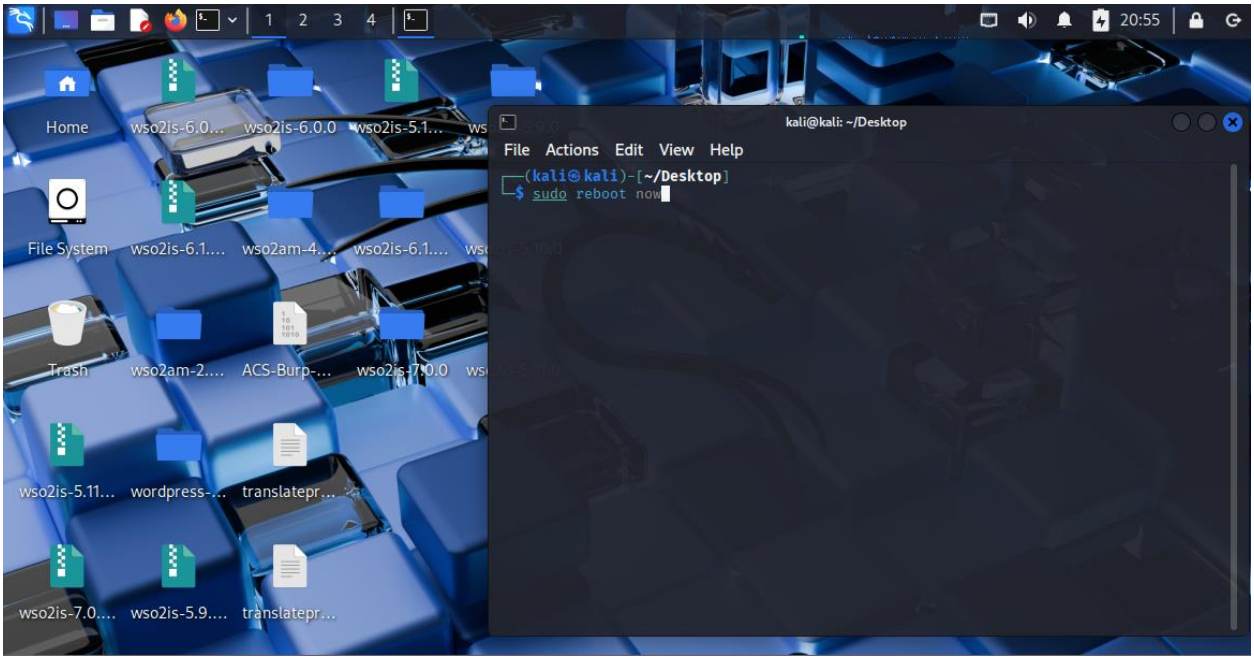


Grant Docker Privileges to the Current User

To allow the current user to run Docker commands without using sudo, add the user to the docker group:

sudo usermod -aG docker $USER

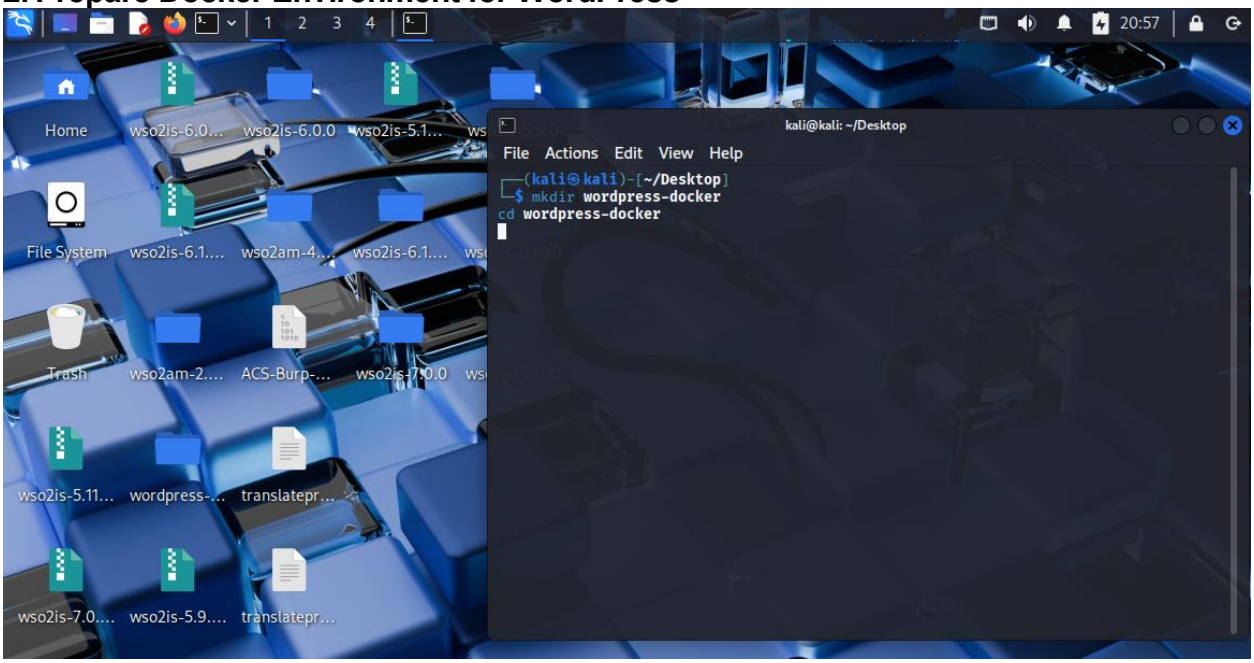
Note: You may need to log out and log back in for this change to take effect.



Reboot the Virtual Machine

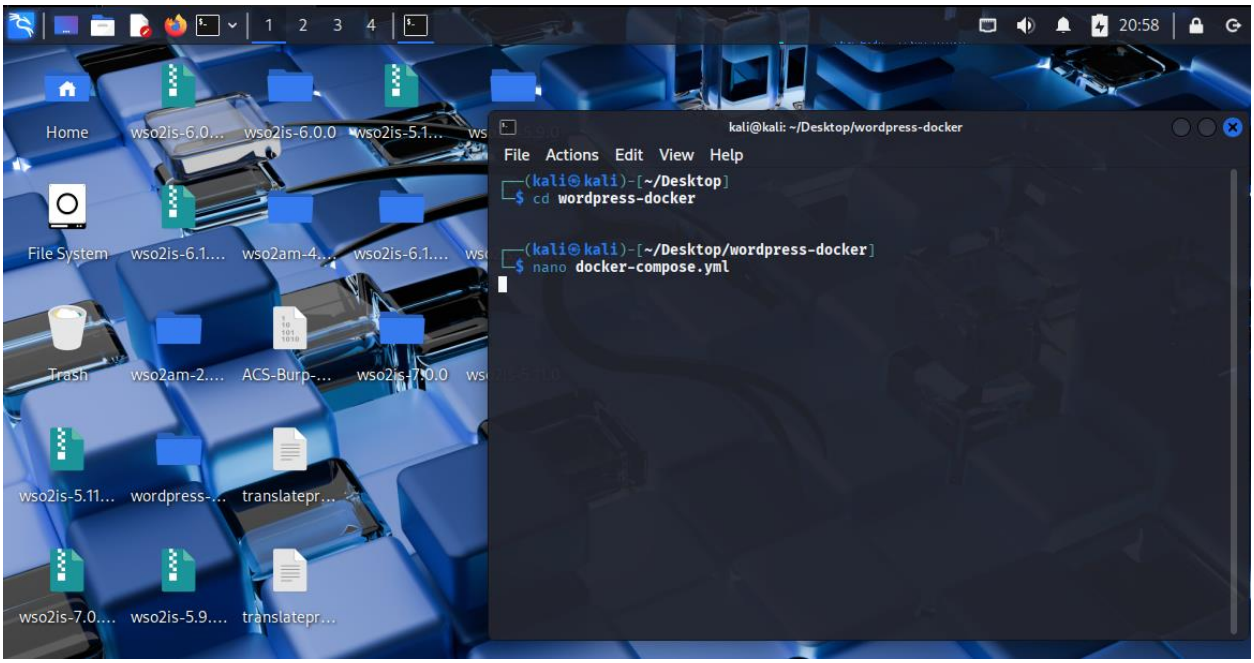
After modifying user group permissions, reboot the system to apply the changes:
sudo reboot now

2. Prepare Docker Environment for WordPress



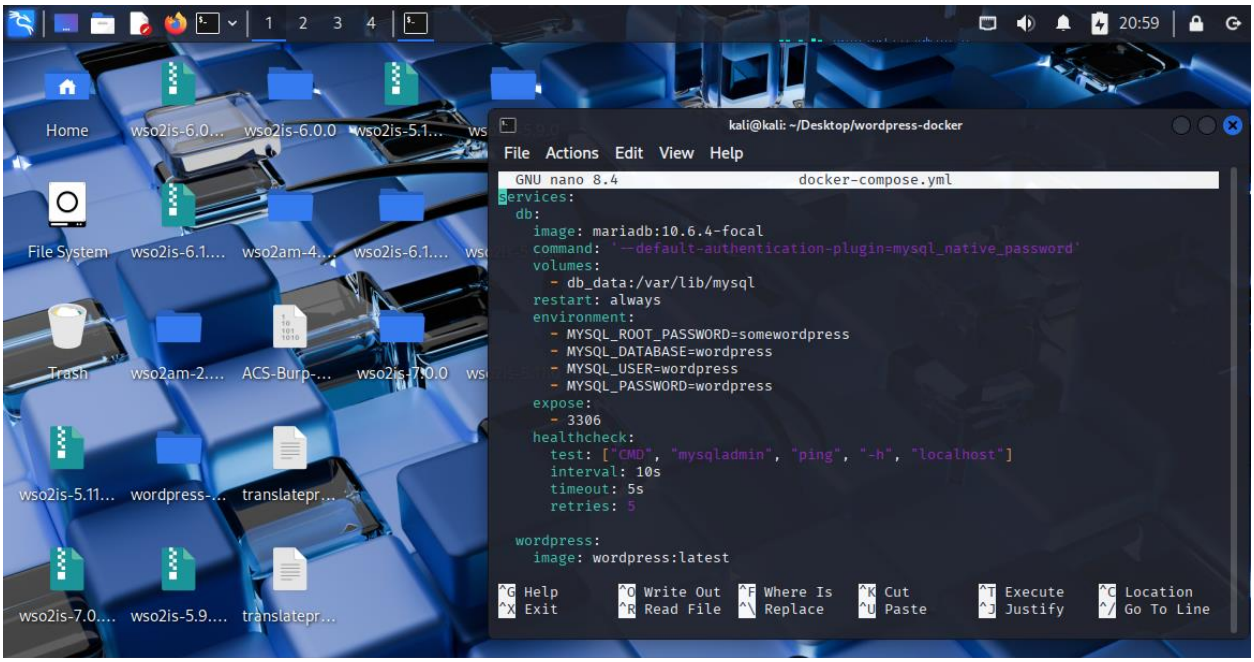
Prepare Docker Environment for WordPress

Open the terminal and create a new directory for the WordPress Docker setup. Then navigate into it using the following commands:
mkdir wordpress-docker
cd wordpress-docker



Create the Docker Compose File

Use a text editor (e.g., nano) to create the Docker Compose configuration file:
nano docker-compose.yml



Add WordPress and Database Configuration to docker-compose.yml

Copy and paste the following content into the docker-compose.yml file:

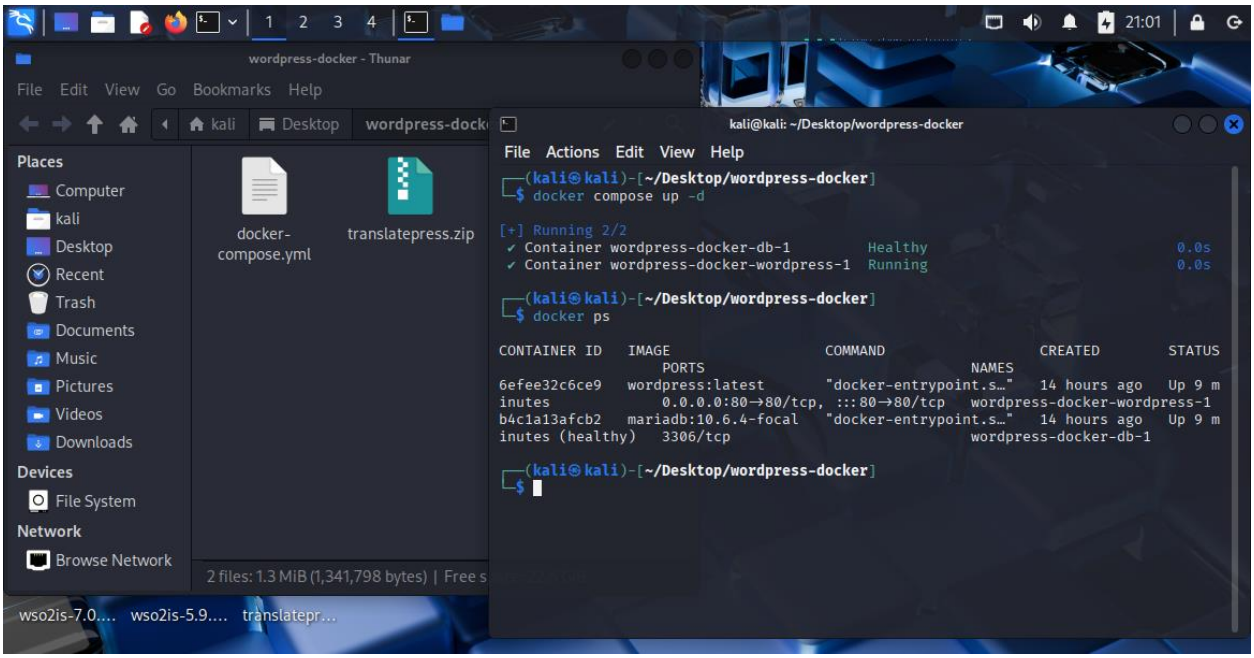
```
services:
  db:
    # We use a mariadb image which supports both amd64 & arm64 architecture
    image: mariadb:10.6.4-focal
    # If you really want to use MySQL, uncomment the following line
    #image: mysql:8.0.27
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:
```

To save and exit:

Press Ctrl + O, then press Enter to save the file

Press Ctrl + X to exit the editor

3. Deploy WordPress Using Docker

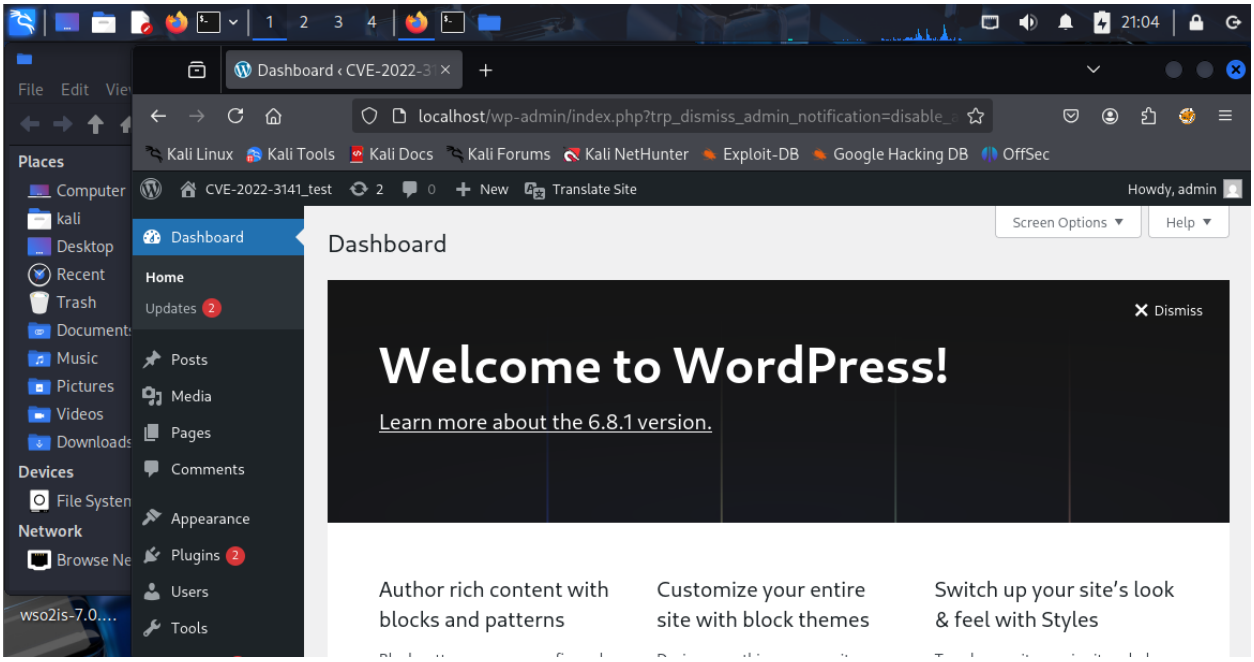


Deploy WordPress Using Docker

Open the terminal and run the following command to start the WordPress and MariaDB containers in detached mode:

docker compose up -d

This command tells Docker Compose to pull the necessary images (if not already available), create the containers, and run them in the background.



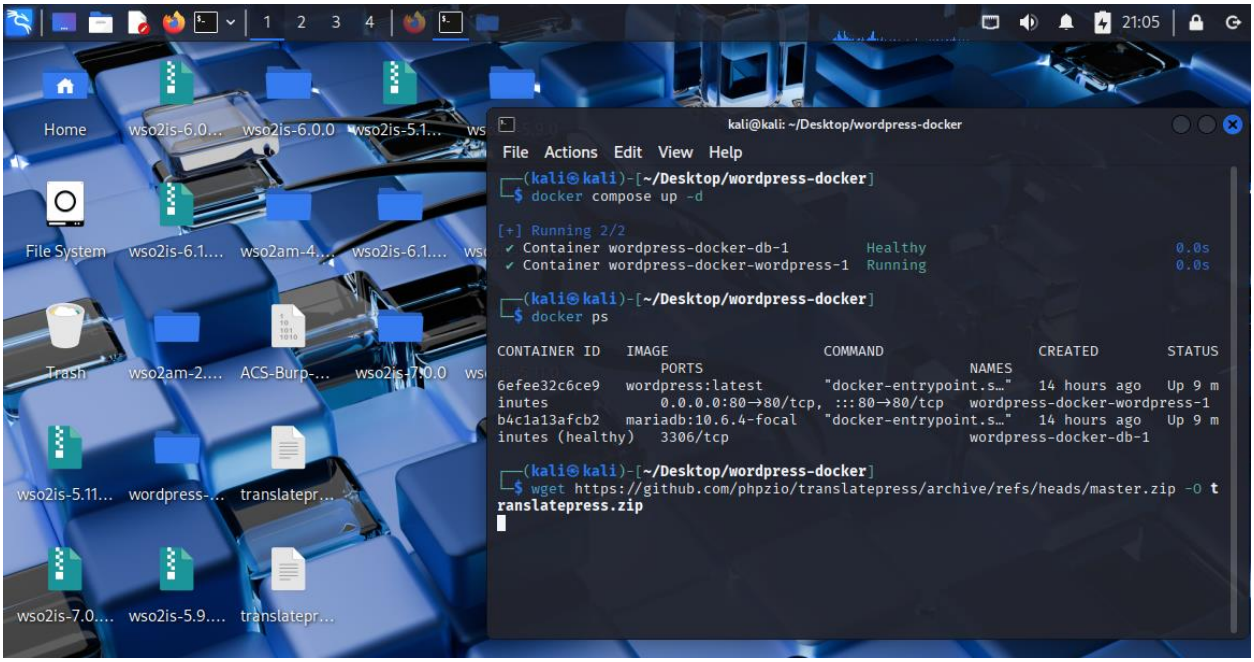
Access WordPress in Your Browser

Once the containers are running, open your web browser and go to the following URL to access the WordPress setup page:

Open your browser and navigate to: http://localhost:80

If you're using a VM, make sure port 80 is properly forwarded or accessible from your host system.

4. Download and Prepare the Vulnerable Plugin



Download and Prepare the Vulnerable Plugin

Open the terminal and run the following command to download the plugin:

wget https://github.com/phpzio/translatepress/archive/refs/heads/master.zip -O translatepress.zip

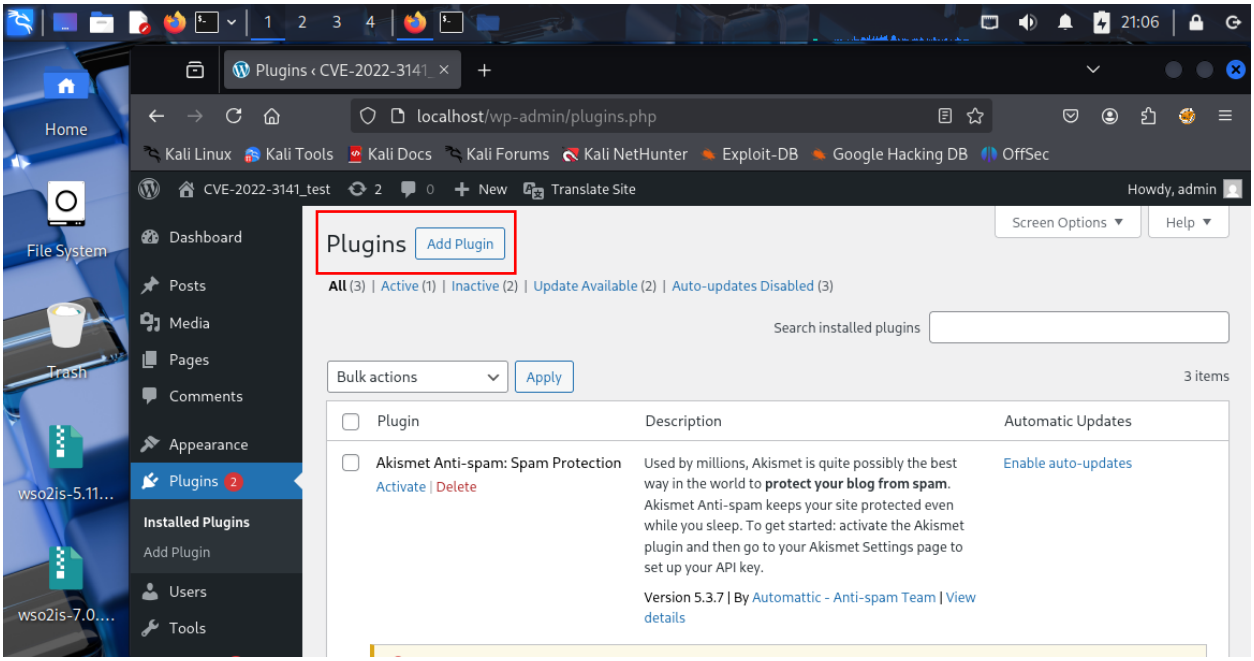
This command downloads the plugin from GitHub and saves it as translatepress.zip.

Install the Plugin in WordPress

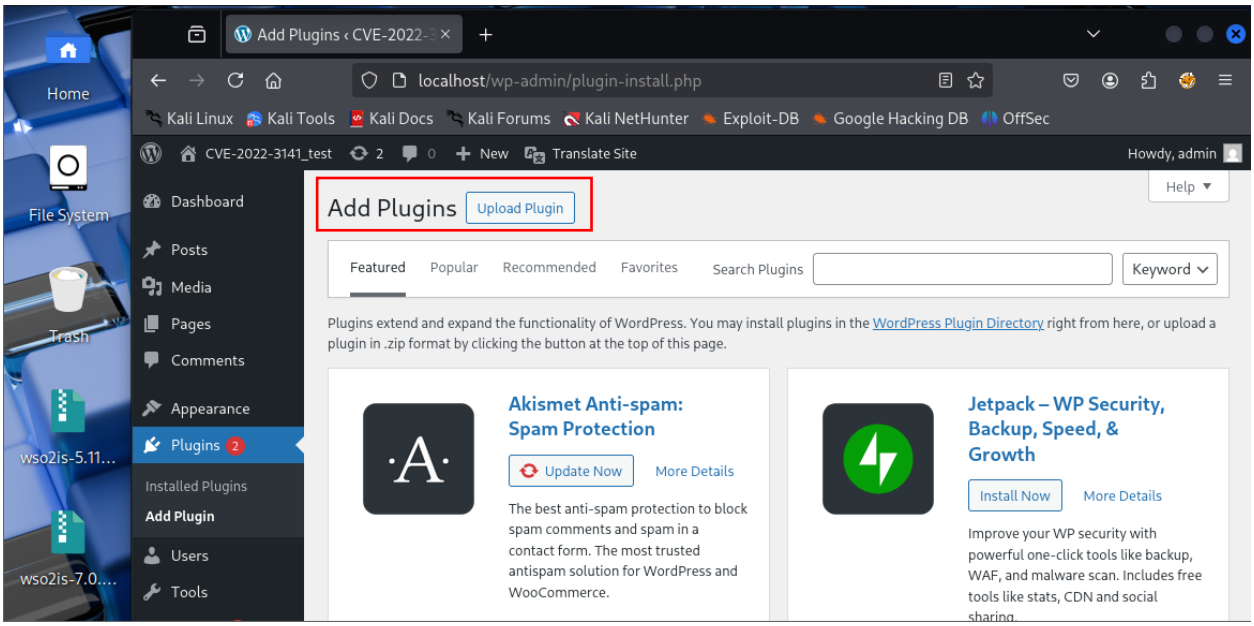
Open your browser and go to the WordPress admin panel:

http://localhost/wp-admin

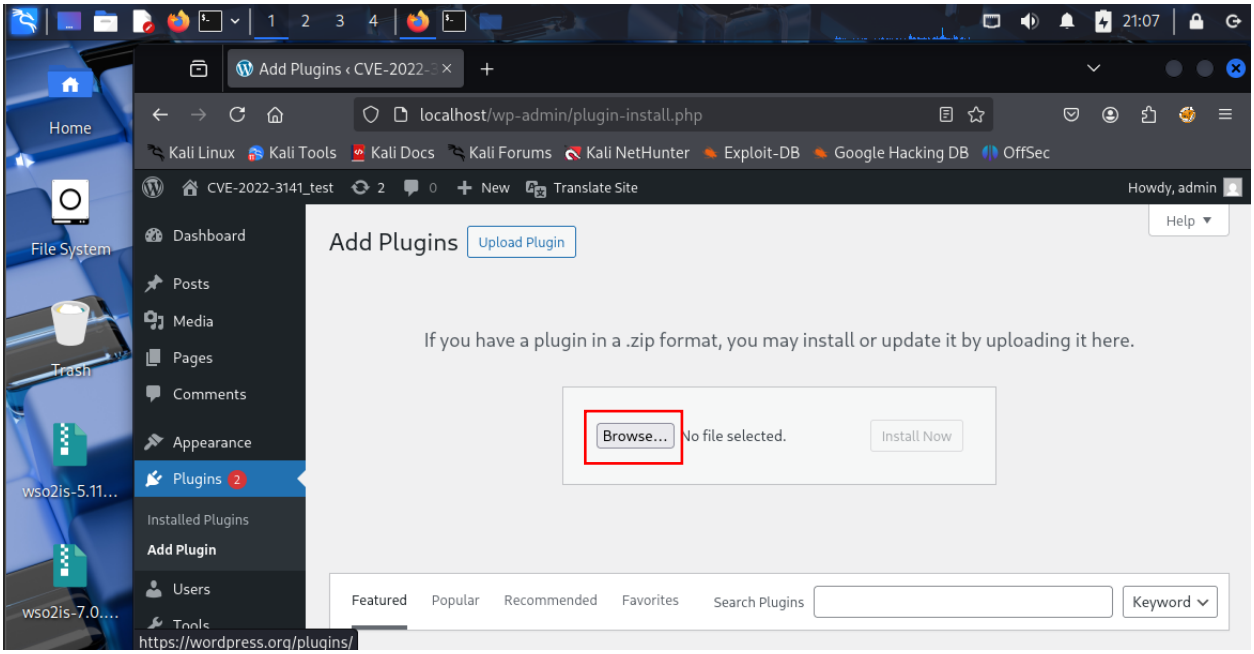
Log in with your admin credentials.

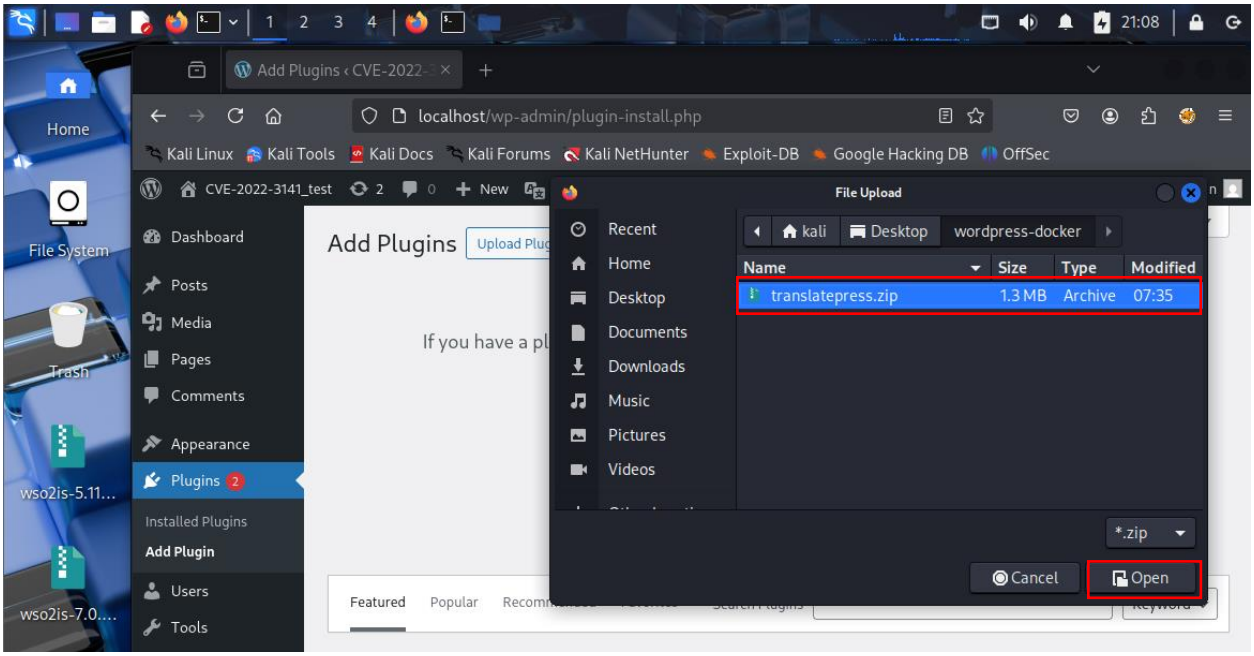


Navigate to Plugins → Add New.

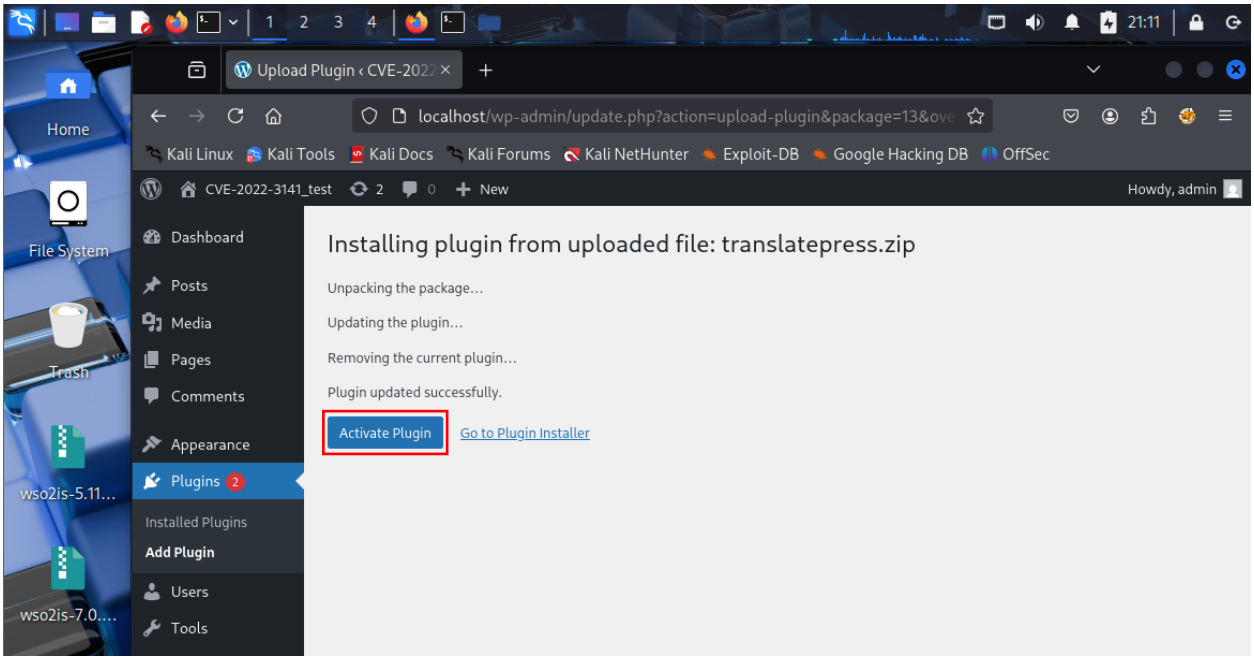


Click on Upload Plugin at the top.





Select the translatepress.zip file that you downloaded earlier.
Click Install Now.



After installation, click Activate Plugin to enable it.

5. Start the Attack

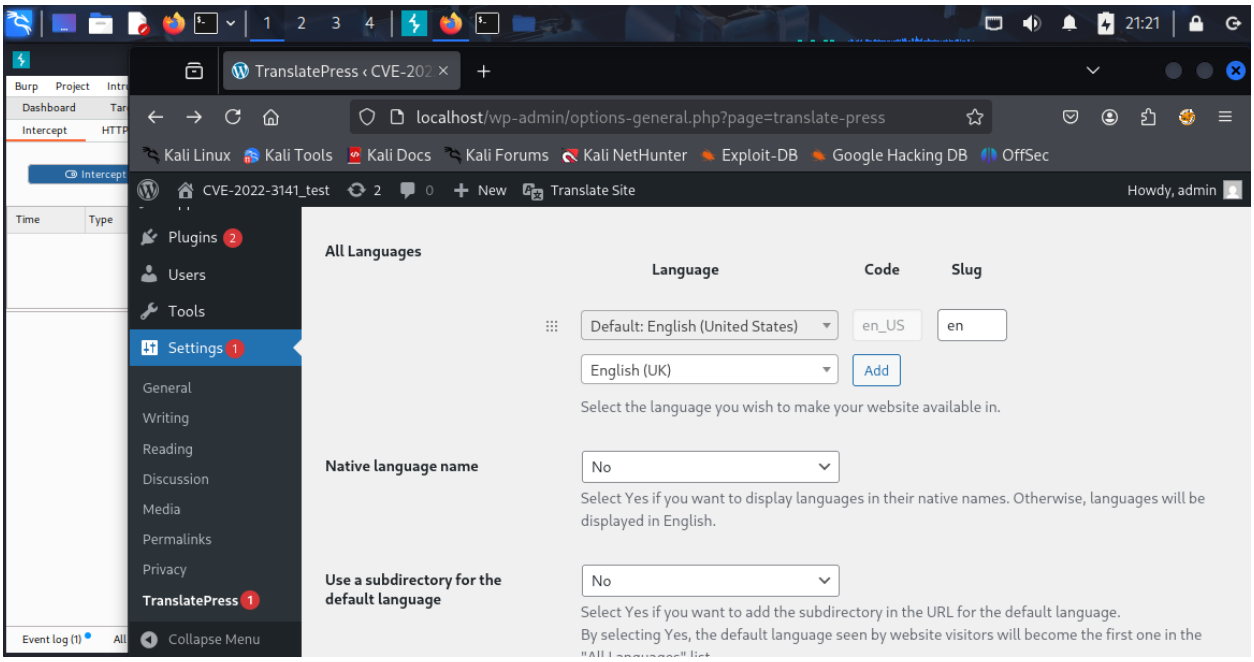
Connect your browser to Burp Suite, log in to Wordpress and add any language from the dropdown (the url to do this is /wp-admin/options-general.php?page=translate-press). In Burp Suite, do a right click→ copy to file, and save it as translatepress_req.txt.

Connect your browser to Burp Suite

Configure your browser to route traffic through Burp Suite (usually via proxy 127.0.0.1:8080).

Log in to the WordPress Admin Panel

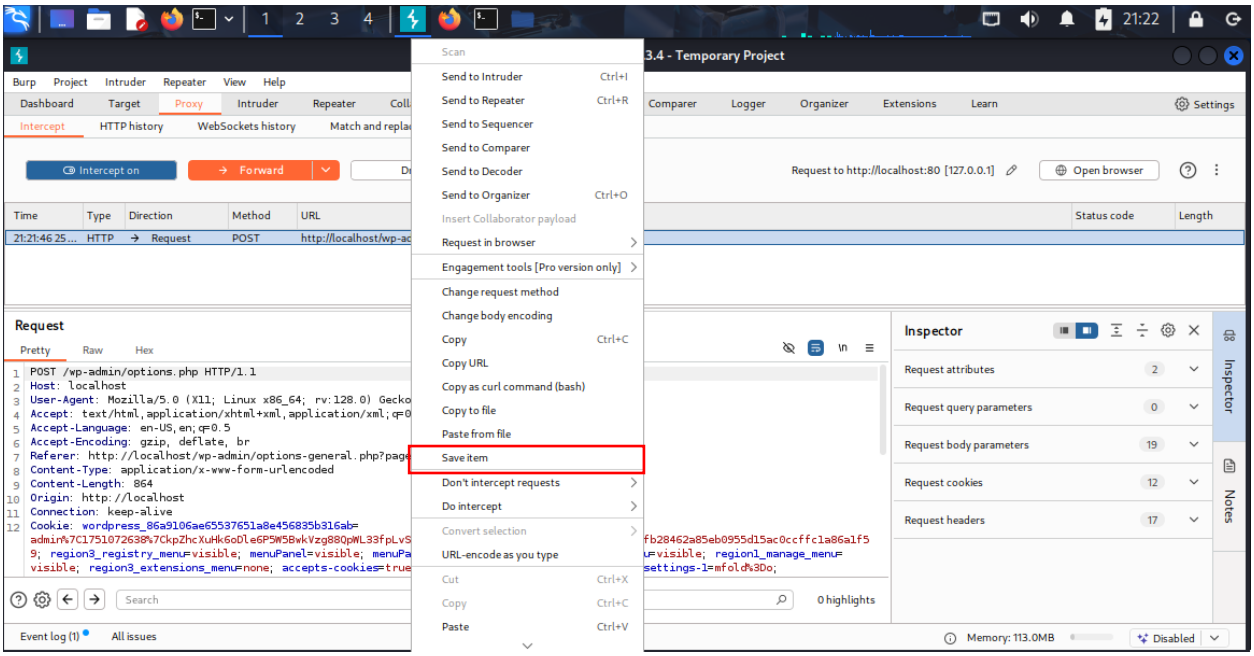
Visit: http://localhost/wp-admin
Enter your admin credentials to log in.

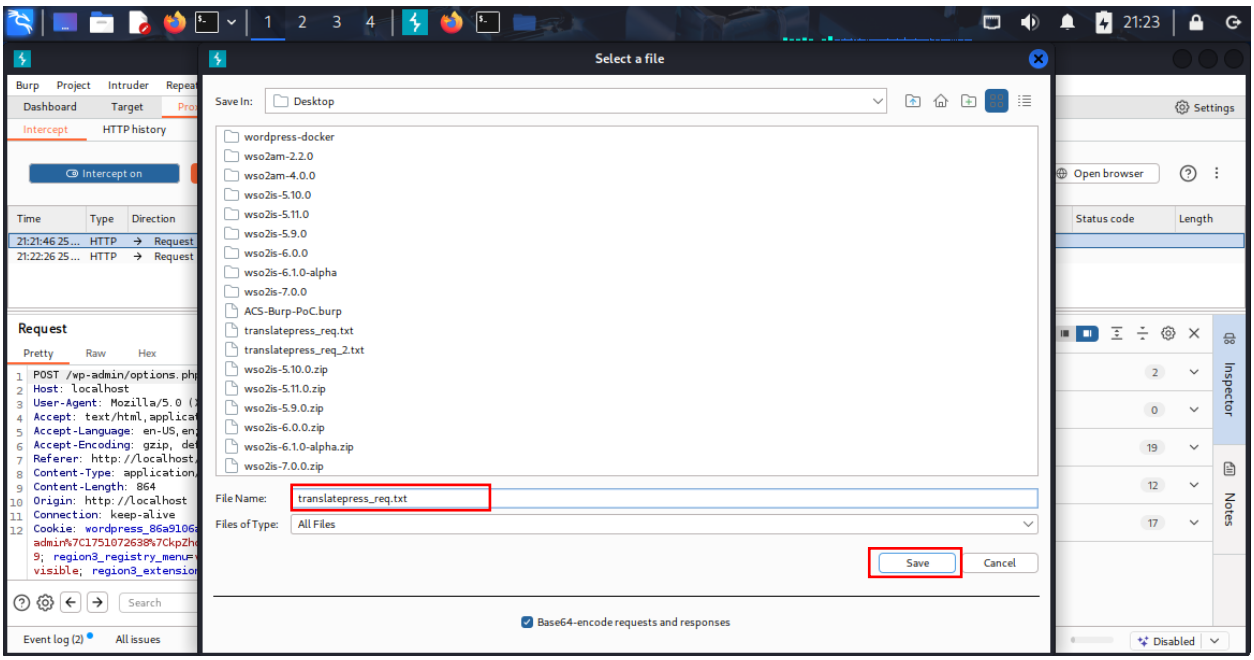


Trigger the Plugin Functionality

Navigate to: /wp-admin/options-general.php?page=translate-press

From the TranslatePress plugin settings page, add any new language using the dropdown.





Capture the Request in Burp Suite

Once the request is sent:

Go to Burp Suite's HTTP history tab.

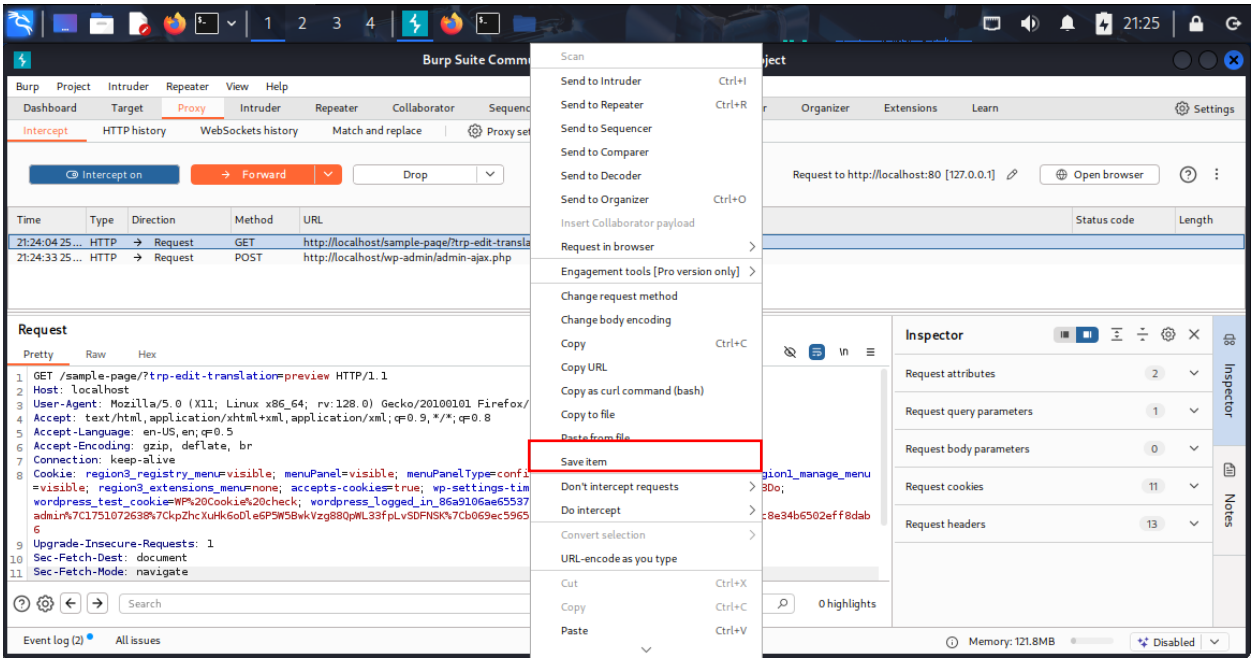
Right-click the captured request.

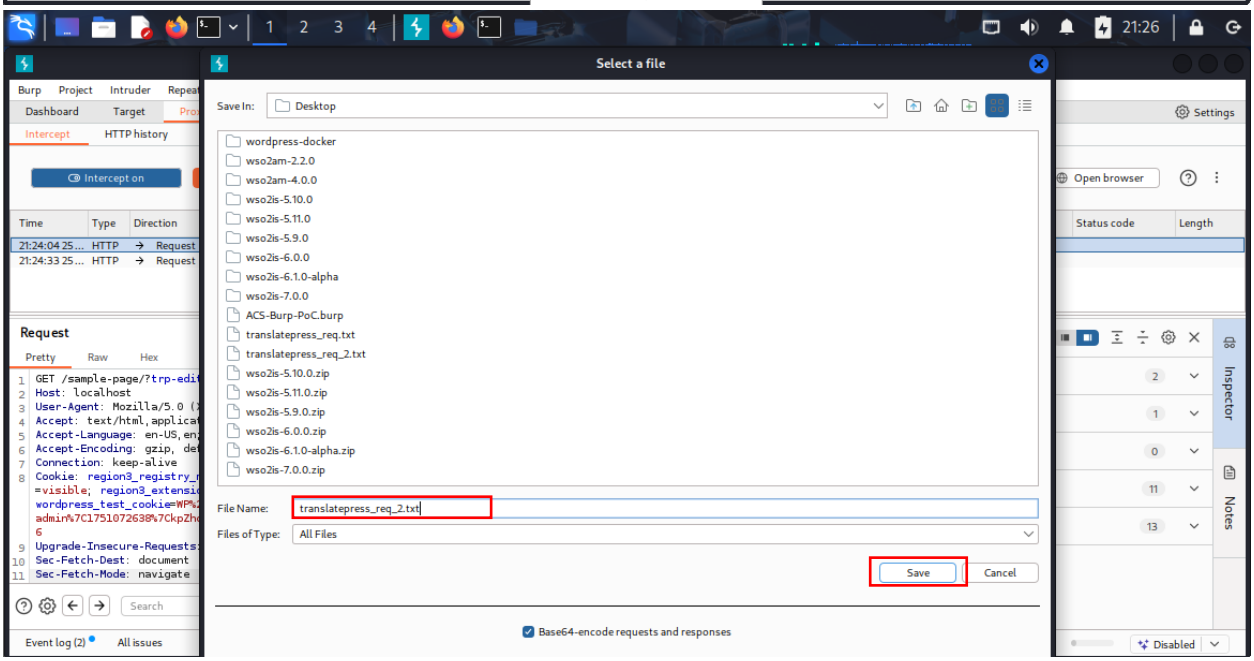
Select "Save item…"

Save it as translatepress_req.txt

Capture Translation Preview Request

In your browser, navigate to the following URL to trigger the translation preview functionality:
http://localhost/sample-page/?trp-edit-translation=preview





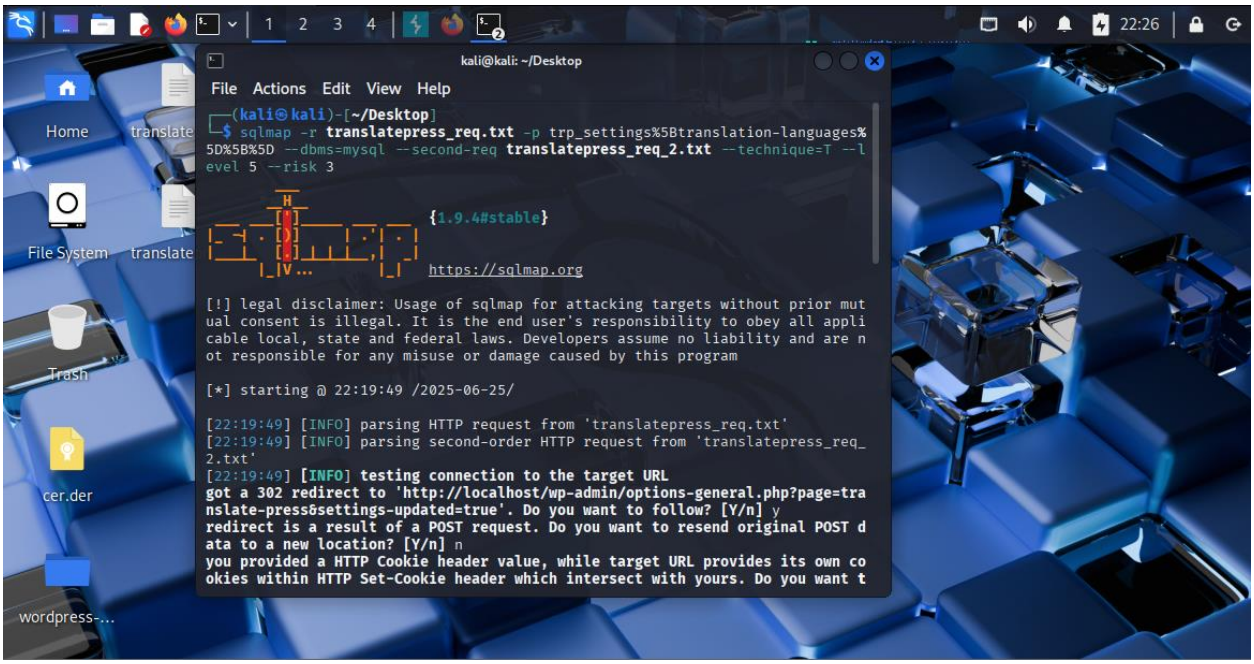
In Burp Suite, locate the request sent to this URL in the HTTP history tab.
Right-click the request, choose "Save item...", and save it as:
translatepress_req_2.txt

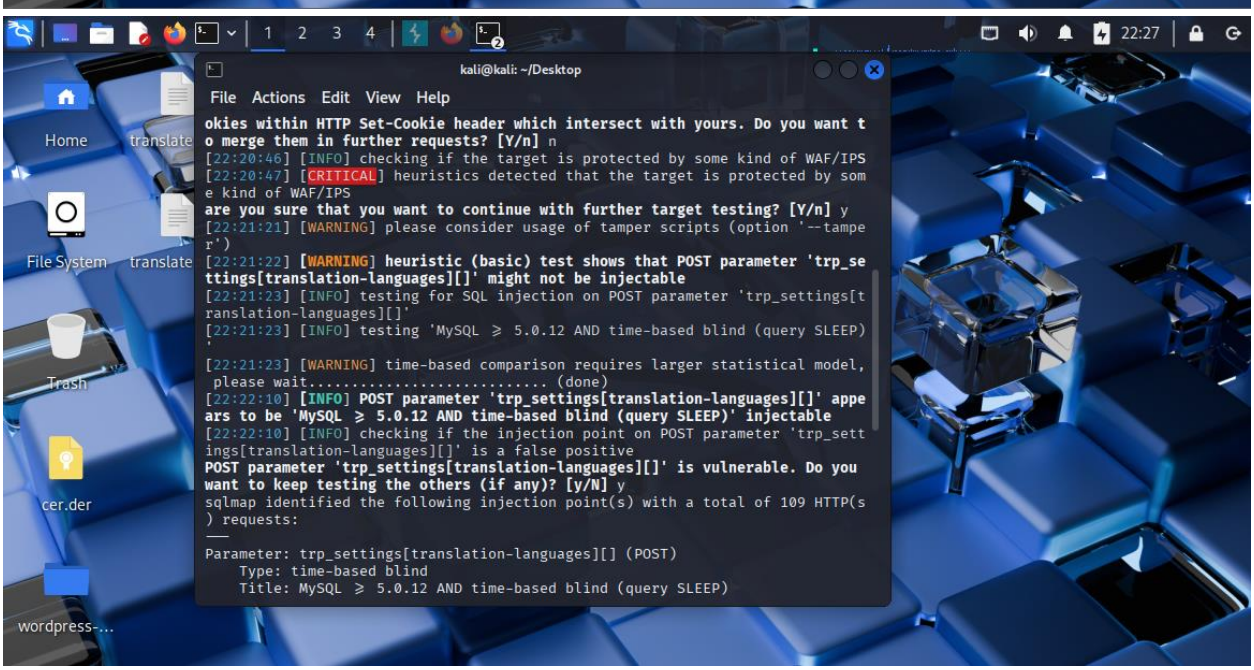
Execute the SQL Injection Attack Using sqlmap

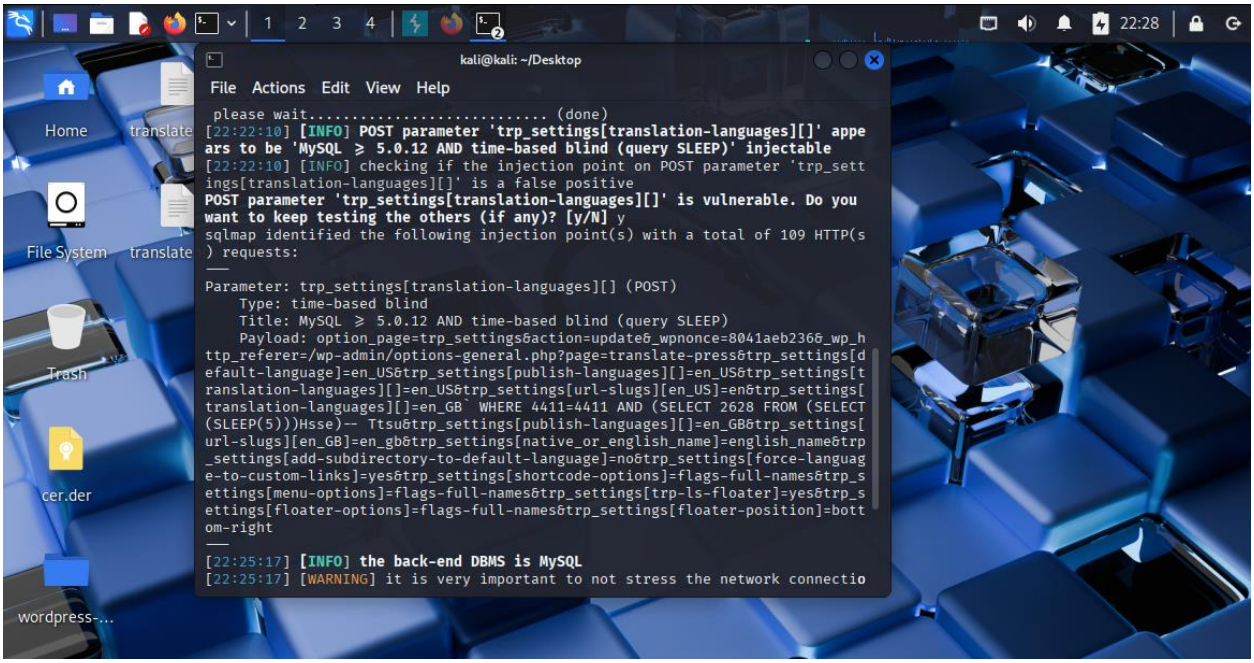
Use the captured requests to perform a time-based blind SQL injection attack against the vulnerable plugin.

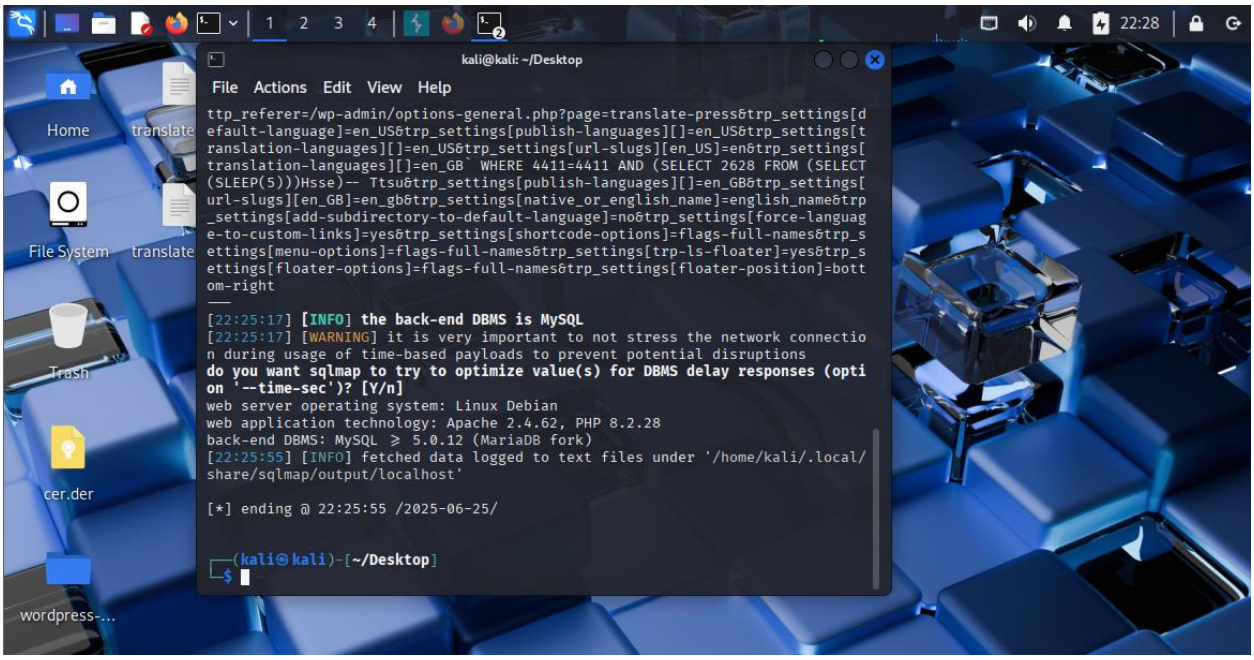
Run the following command in the terminal:

sqlmap -r translatepress_req.txt -p trp_settings%5Btranslation-languages%5D%5B%5D --dbms=mysql --second-req translatepress_req_2.txt --technique=T --level 5 --risk 3





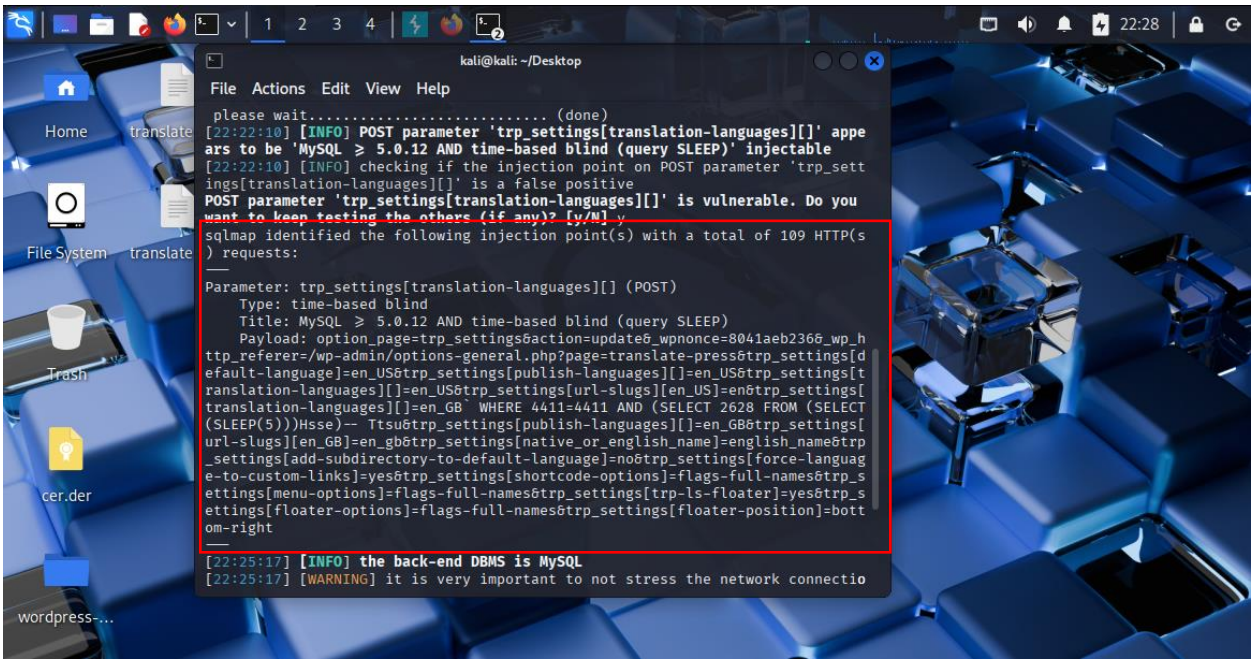




Attack Results Summary

The security assessment using sqlmap confirmed a time-based blind SQL injection vulnerability in the TranslatePress plugin for WordPress, specifically in the trp_settings[translation-languages][] POST parameter. The exploit was successfully demonstrated through a series of HTTP requests captured and analyzed via Burp Suite.

Confirmed Injection



Conclusion

Summarize that CVE-2022-3141 is a critical vulnerability that can be exploited by authenticated users to perform SQL injection and compromise WordPress data. Time-based blind injection was confirmed as effective using sqlmap.

Mitigation / Recommendations

- Upgrade TranslatePress to version 2.3.3 or higher
- Restrict plugin configuration access to trusted users only
- Monitor for suspicious activity via WordPress logs

References

https://www.cve.org/CVERecord?id=CVE-2022-3141
https://wpscan.com/vulnerability/1fa355d1-cca8-4b27-9d21-0b420a2e1bf3/
https://nvd.nist.gov/vuln/detail/CVE-2022-3141
https://medium.com/@elias.hohl/authenticated-sql-injection-vulnerability-in-translatepress-multilingual-wordpress-plugin-effc08eda514
https://github.com/ehtec/translatepress-exploit