

## PHP 練習問題. 練習 14 脆弱性対策

「脆弱性」とは、攻撃の対象になりうる「不具合」のことです。

### 設問1. SQL インジェクション対策

「練習問題 08-4」で、レコードを数件入力してください。

TODOリスト

2020/03/17

TODO項目を入力してください

追加

期限日	TODO項目
2020-03-05	テスト 1
2020-03-06	テスト 2
2020-03-07	テスト 3
2020-03-08	テスト 4
2020-03-09	テスト 5

「練習問題 08-4」とは別のディレクトリに index.php に下記のフォームを作成してください。form の action は show.php にしてください。

- index.php

TODOリストの期限日で検索

検索日

2020/03/09

検索

show.php では「練習問題 08-4」と同じデータベースに接続します。  
index.php で入力した日付で期限日を検索し、一致したレコードを一覧表示します。

ただし、SQL 文を組み立てるときにパラメータを使わず、POST されてきた値を直接 SQL 文に挿入してください。

例)

```
$sql = "select * from todo_items ";  
$sql .= "where ";  
$sql .= "expiration_date='" . $_POST['date'] . "'";  
$stmt = $dbh->prepare($sql);  
$stmt->execute();
```

index.php でテキストボックスに日付を入力し、show.php で該当の期限日のレコードが表示されることを確認してください。

- show.php

期限日	TODO項目
2020-03-09	テスト5

次に、index.php で、日付の代わりに下記の文字列を入力し、送信してください。

**' ' OR 1=1 -- '**

- inedx.php

TODOリストの期限日で検索

検索日

検索

show.php で表示される内容を確認してください。

- show.php

期限日	TODO項目
2020-03-05	テスト 1
2020-03-06	テスト 2
2020-03-07	テスト 3
2020-03-08	テスト 4
2020-03-09	テスト 5

テーブルに保存されているレコードが全件表示されました。  
**なぜそのような表示になるか、説明してください。**

(ヒント)  
SQL 文を `var_dump()` でダンプしてみましょう。

(解説)

検索条件やデータの登録のために文字列を入力するテキストボックスに、SQL 文として意味のある文字列を入力することで、思わぬ動作をすることがあります。

- ✓ ログイン情報が一致していないのにログインできてしまう。
- ✓ ログインユーザーのものしか表示できないはずのプロファイルが、全ユーザーのプロファイルが表示されてしまう。

悪意を持った第三者に利用されることで、個人情報や機密情報が外部に漏れる原因になります。

このような不具合を利用した攻撃のことを SQL インジェクションといいます。

PHP で SQL インジェクション対策を行うには、データベースの処理に PDO クラスを使用し、SQL 文にはパラメータを埋め込み、bindValue()などのメソッドを使うことです。

例)

```
$sql = "select * ";  
$sql .= "from todo_items ";  
$sql .= "where expiration_date=:date";  
$stmt = $dbh->prepare($sql);  
$stmt->bindValue(':date', $_POST['date'], PDO::PARAM_STR);  
$stmt->execute();
```

「練習問題 08」の解答では、SQL 文にパラメータを使用し、bindValue()を使っているため、SQL インジェクションは発生しません。

「練習問題 08」以降で、POST や GET など、プログラムの外部から取得した値を直接 SQL 文に挿入して使っている箇所があれば、SQL インジェクションの脆弱性があります。

※ PHP の外部からやってくるデータを信用してはいけません。

## 設問2.XSS（クロスサイトスクリプティング）対策

「練習問題 08-4」で、TODO 項目に下記の文字列を入力して「追加」をクリックしてください。

**<script>while(1){alert('test')}</script>**

どのような挙動になるでしょうか？また、なぜ、**そのような挙動になるか、説明してください。**

The screenshot shows a web application interface. At the top, there is a header labeled 'TODOリスト'. Below it, there is a form with two input fields and a button. The first input field contains the date '2020/03/10'. The second input field contains the JavaScript code '<script>while(true){alert('test')}</script>'. To the right of the second input field is a blue button labeled '追加'. Below the form, there is a text area that says '「OK」をクリックしても、何度もダイアログが表示される。'. At the bottom, there is a modal dialog box titled 'localhost の内容'. Inside the dialog, it says 'test'. There is a blue button labeled 'OK' at the bottom right of the dialog.

(解説)

このような不具合を利用した攻撃手法のことを **XSS（クロス・サイト・スクリプティング）** といいます。

「設問 2」の事例では、「OK」ボタンをクリックしても永久にダイアログが開き続けるだけです。しかし、悪意を持った第三者が巧妙に作成されたスクリプトを投稿することで、閲覧者の情報を抜き取ったり、他のサイトを攻撃したりすることが可能になります。

XSS は、受信した値を **htmlspecialchars()**関数を使用し、スクリプトとして意味のある文字を別の文字に変換することで回避することができます。

**htmlspecialchars** — 特殊文字を HTML エンティティに変換する

<https://www.php.net/manual/ja/function.htmlspecialchars>

このような処理のことを**サニタイズ**といいます。サニタイズを行うことで、**XSS 対策**を施すことができます。

※ PHP の外部からやってくるデータを信用してはいけません。

「練習問題 08-4」の show.php を改良してサニタイズ処理を追加し、XSS 攻撃に対処できるようにしてください。

TODOリスト

期限日	TODO項目
2020-03-05	テスト 1
2020-03-06	テスト 2
2020-03-07	テスト 3
2020-03-08	テスト 4
2020-03-09	テスト 5

TODOリスト

期限日	TODO項目
2020-03-05	テスト 1
2020-03-06	テスト 2
2020-03-07	テスト 3
2020-03-08	テスト 4
2020-03-09	テスト 5
2020-03-17	<script>while(true){alert('test')}</script>

対処の前後で、phpMyAdmin などのツールを使い、レコードにどのように保存されているか比較してください。

また、改良後にブラウザで「ページのソースを表示」を行い、HTML がどのように出力されているか確認してください。

### 設問3. CSRF（クロスサイトリクエストフォージェリ）対策

「練習問題 08-4」をブラウザで表示します。

TODOリスト

2020/03/19

TODO項目を入力してください

追加

マスの右ボタンをクリックし、「ページのソースを表示」します。

```
1 <!DOCTYPE html>
2 <html lang="jp">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>練習問題08-4</title>
8   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-
9   Vko08x4CGs03+Hhvx8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
10  <style>
11    #expiration_date {
12      width: 12rem;
13    }
14    #todo_item {
15      width: 25rem;
16    }
17  </style>
18 </head>
19
20 <body>
21   <div class="container">
22     <div class="row my-3">
23       <div class="col-md-2"></div>
24       <div class="col-md-8">
25         <div class="card">
26           <div class="card-header">TODOリスト</div>
27           <div class="card-body">
28             <form action="./add.php" method="post" class="form-inline">
29               <div class="form-group mb-3 mr-1">
30                 <label for="expiration_date" class="sr-only">期限日</label>
31                 <input type="date" name="expiration_date" value="2020-03-19" id="expiration_date" class="form-control">
32               </div>
33             </form>
34           </div>
35         </div>
36       </div>
37     </div>
38   </div>
39 </body>
40 </html>
```

表示された HTML のソースを全て選択し、テキストエディタに貼り付けます。ローカルに（例えば、デスクトップやドキュメントに test.html などと名前をつけて）HTML として保存してください。

HTML として保存したファイルを編集します。

form の action を下記のように、XAMPP（または MAMP）のサーバー上の URL に対して送信されるように編集して保存してください。（編集後の URL は、皆さんの環境に合わせてください）

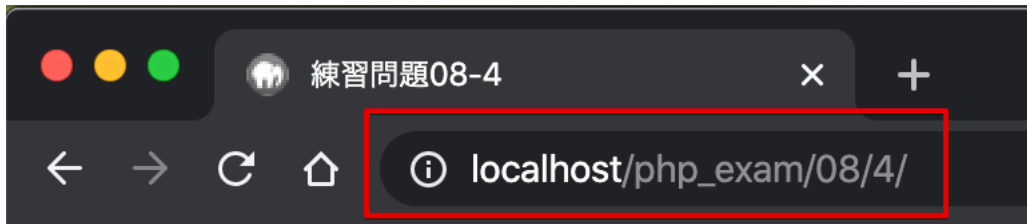
編集前)

```
<form action="./add.php" method="post">
```

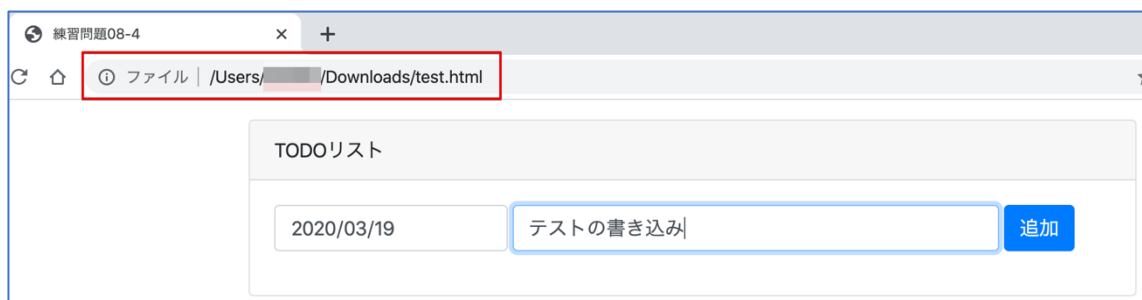
編集後)

```
<form action="http://localhost/php_exam/08/4/add.php"
method="post">
```

編集後の form の action の URL は、ブラウザに表示されている URL を参考にしてください。



ローカルに保存した HTML ファイルをブラウザで開いてください。



↑ ローカルのファイルです。

ローカルのファイルから項目を入力し、「追加」ボタンをクリックしてください。



上記のように、サーバー外のフォームからサーバーに対してデータを送信できてしまうことを確認してください。



解説)

一見、何の問題もないように思われるかもしれませんが、下記のような危険性をはらんでいます。

- (1) 悪意を持った第三者が攻撃用の Web ページを作成し、ユーザーを攻撃用の Web ページに誘導し、不正なデータをサーバーに対して送信させます。
- (2) 攻撃を受けたサーバーは、意図しない処理を行ってしまいます。
- (3) 攻撃者は攻撃を行うサーバーへアクセスすることなく、攻撃対象の Web アプリケーションに対して攻撃を行うことができます。

その結果、

- ✓ いたずらや犯罪予告、不正サイトへの誘導など。
- ✓ 大量に不正な書き込みを行うことで、サーバーの能力を超える処理を行わせ、サーバーをダウンさせる。(DoS 攻撃、DDoS 攻撃)

ことなどが可能になってしまいます。

このような攻撃手法のことを **CSRF (クロス・サイト・リクエスト・フォージェリ)** と言います。

※ *PHP の外部からやってくるデータを信用してはいけません。*

**CSRF** に対処するにはいくつか方法があります。

- (1) 送信元の URL を確認する。

下記の変数を使って、送信元の URL を確認することができます。

**`$_SERVER['HTTP_REFERER']`**

※ `phpinfo()` という関数で、サーバーの様々な情報を知ることができます。

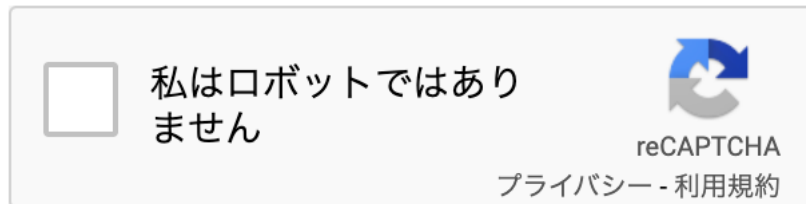
送信元のサーバーのホスト名が、送信先のサーバーのホスト名と一致したときのみ、プログラムの処理を行うようにします。

しかし、`HTTP_REFERER` の値はブラウザが保持しており、リクエストの際にブラウザがサーバーに対して送信してくるものなので、簡単に改ざんすることが可能です。また、セキュリティソフトによっては `HTTP_REFERER` を送らないようにするものもあります。

→最近ではあまり使わない手法です。

(2) 画像認証を使う。

下記のような仕組みをご覧になったことがないでしょうか？



チェックボックスにチェックを入れさせたり、画像に表示された崩れた文字を入力させることで、送信元の URL を確認するものです。PHP でも利用することは可能です。

しかし、**ユーザーに不便を強いるというユーザビリティの観点**から嫌われることが多いです。

また、**AI を使った画像認識の精度が向上**したことで簡単に突破されてしまったという事例もあって、必ずしも安全とは言えないようです。

(3) ワンタイムトークンを利用する。

入力フォームに「1 回のアクセスだけ有効な文字列」を<input type="hidden">を使って埋め込みます。送信先では、その文字列が正しいかどうかを判断して、正しいときのみ処理を行います。そのような文字列のことを**ワンタイムトークン**といいます。

**ワンタイムトークン**は「乱数」を使って発生させ、\$\_SESSION に保存します。<input type="hidden">に埋め込まれた文字列と\$\_SESSION に保存された文字列を比較することで、実装ができます。

複雑な乱数を発生させ、かつ、強度の高い暗号化処理を行うこと、アクセスされるたびに別の文字列に更新することで、安全性を高めることができます。

「設問 3」では、実装が簡単で安全性が高い**ワンタイムトークン**を使って**CSRF 対策**を試みます。

- 送信側のフォームの実装

```
// ワンタイムトークンを生成してセッションに保存します。  
$token = bin2hex(openssl_random_pseudo_bytes(32));  
$_SESSION['token'] = $token;
```

```
<!--
```

セッションに保存したワンタイムトークンを hidden で POST します

```
-->
```

```
<input type="hidden" name="token" value="<?= $token ?>">
```

※ 実際にどのような文字列が発生しているか確認してみましょう。

※ また、リロードするたびに文字列が変更されることを確認してください。

- 受信側の実装

```
// フォームで送信されてきたトークンが正しいかどうか確認(CSRF 対策)  
if (!isset($_SESSION['token']) || $_SESSION['token'] !== $_POST['token']) {  
    $_SESSION['err_msg'] = "不正な処理が行われました。";  
    header('Location: ./');  
    exit;  
}
```

- では、実際に「練習問題 08-4」に CSRF 対策を施してください。

✓ 正常な動作

TODOリスト

2020/03/19

ですと1

追加

TODOリスト

2020/03/19

TODO項目を入力してください

追加

期限日	TODO項目
2020-03-19	テスト 1

正常な動作では、項目を追加することができます。

✓ CSRF を検知したとき（サーバー外のフォームから POST されたとき）

TODOリスト

不正な処理が行われました。

2020/03/19

TODO項目を入力してください

追加

期限日	TODO項目
2020-03-19	テスト 1

CSRF 攻撃を検知したときは、エラーメッセージが表示されます。

設問4. 「練習問題 13-8」をコピーして使用してください。

ソースに、下記の対策を施してください。

- ✓ SQL インジェクション対策
- ✓ XSS 対策
- ✓ CSRF 対策

CSRF 攻撃を検知したときは、「設問 3」のように、送信元と同じフォームにエラーメッセージを表示してください。(全ての送信フォームに必要です)

- サーバー外のフォームからの送信

TODOリスト

ログインユーザー：さとみ

ログアウト

2020/03/21

TODO項目を入力してください

追加

期限日	TODO項目
2020-03-20	テスト 2

☐ 未完了

☒ 完了

☐ 削除

実行

CSVファイルに変換

CSVファイルをダウンロード

CSVファイルをアップロードして更新

上記の場合は、「完了」を選択して「実行」ボタンをクリックしました。

- CSRF 攻撃を察知したときは、エラーメッセージを表示します。

TODOリスト

ログインユーザー：さとみ

ログアウト

不正な処理が行われました。

2020/03/21

TODO項目を入力してください

追加

期限日	TODO項目
2020-03-20	テスト 2

☒ 未完了

☐ 完了

☐ 削除

実行

CSVファイルに変換

CSVファイルをダウンロード

CSVファイルをアップロードして更新

設問5. 「設問4」に加えて、例外が発生したときの処理を追加します。

これまでの解答では、例外発生時には `var_dump()` を用いて、例外の内容を画面に表示していました。

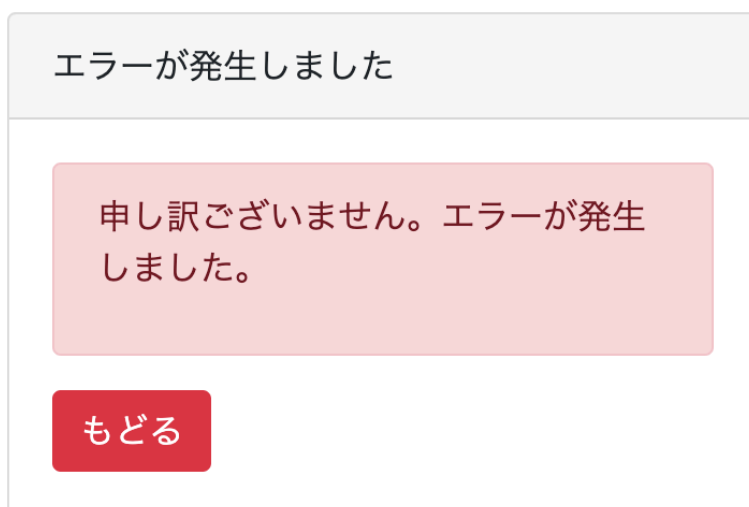
```
object(PDOException)#4 (8) { ["message":protected]=> string(48) "SQLSTATE[HY000] [2002] No such file or
directory" ["string":"Exception":private]=> string(0) "" ["code":protected]=> int(2002) ["file":protected]=>
string(105) "/Users/ /14/4/class/db/Base.php"
["line":protected]=> int(20) ["trace":"Exception":private]=> array(3) { [0]=> array(6) { ["file"]=> string(105)
"/Users/ s/db/Base.php" ["line"]=>
int(20) ["function"]=> string(11) "__construct" ["class"]=> string(3) "PDO" ["type"]=> string(2) "->" ["args"]=>
array(3) { [0]=> string(49) "mysql dbname=php_work;host=localhost;charset=utf8" [1]=> string(4) "root" [2]=>
string(4) "root" } [1]=> array(6) { ["file"]=> string(110) "/Users/
/14/4/class/db/Todoltems.php" ["line"]=> int(14) ["function"]=> string(11) "__construct"
["class"]=> string(4) "Base" ["type"]=> string(2) "->" ["args"]=> array(0) { } } [2]=> array(6) { ["file"]=> string(97)
"/Users/ /index.php" ["line"]=> int(30)
["function"]=> string(11) "__construct" ["class"]=> string(9) "Todoltems" ["type"]=> string(2) "->" ["args"]=>
array(0) { } } ["previous":"Exception":private]=> NULL ["errorInfo"]=> NULL }
```

開発途中では、エラーの内容を知るために画面に表示して確認することが必要ですが、実運用の段階になると不要です。また、例外の内容や発生箇所を第三者に知られることになってしまいます。

画面に表示された内容を元に、悪意を持った第三者に脆弱性を発見され、攻撃を受ける可能性もあります。

そこで、例外が発生したときには、エラー表示専用のページへリダイレクトし、そちらのページにエラーメッセージを表示するようにしてください。

- error.php



「もどる」(ボタンではなくリンクで結構です)をクリックすると、ログアウト処理のページにリンクするようにしてください。

## 設問6. ブルートフォースアタック対策

**ブルートフォースアタック（総当たり攻撃）**とは、ログインフォームなどに対して、何度もユーザーアカウントを入力し、ログインを行おうという攻撃手法です。

人の手によって行われるよりも、bot と呼ばれるコンピューターのプログラムによって行われることが多いです。

ログインユーザー名とパスワードは「辞書」と呼ばれる「よく使われやすい文字列」から自動的に選ばれたり、ランダムな文字列を生成して使われたりします。

もし、ブルートフォースアタックでログインされてしまったら、Web アプリケーション上のログインユーザーの権限を悪意を持った第三者に奪われることになります。

ブルートフォースアタック自体は、Web アプリケーションの脆弱性を利用された攻撃ではありません。しかし、何らかの対処をしておく必要があります。

ブルートフォースアタックへの対処方法としては、一定回数以上のログイン失敗を繰り返すと、一定時間ログインをできなくする、という方法が考えられます。例えば、セッションにログイン試行回数を保存しておくということもひとつの対策です。

では、設問です。

これまでの脆弱性対策に追加して、ブルートフォースアタック対策を施してください。3 回ログインに失敗すると、セッションがタイムアウト（無効になる）まで、ログインできないようにします。

（セッションは PHP の設定によりますが、数十分で無効になります）

- login.php

ログイン

Email address

Password

ログイン

ログインに 3 回失敗し、4 回目にログインしようとする、

- error.php

エラーが発生しました

ログインできません

もどる

error.php にリダイレクトし、「ログインできません」と表示します。

「戻る」をクリックするとログインページに戻りますが、再度ログインしようすると error.php にリダイレクトします。

セッションが無効になったあと、もしくは、いったんブラウザを閉じてからであればセッションがなくなりますので、再度ログインできるようになります。