

Webアプリ動作環境

...

PHP(mod_php)

Apache + PHP(mod_php) +
MySQL/SQLite

サービス構成

PHPで動作するWebアプリのサービス構成は

- Webサーバー：Apache
- Webアプリ動作方法(アプリケーション・サーバー)：mod_php
- データベース：MySQL or SQLite

これらのサービスをDockerのコンテナで動作させます。データベースはMySQLもしくはSQLiteを使用します。

【注意点】

コンテナ間で接続する場合はIPアドレスでなく、サービス名かコンテナ名で接続します。MySQLとPHPで接続を行う場合のホスト名は、localhostではなくサービス名かコンテナ名で接続します。

ファイル構成

環境構築のためのファイル構成は次のようになります。

```
docker-lamp          // 開発環境ディレクトリ
├── compose.yaml      // Dockerイメージとコンテナを生成・起動するための情報
├── htdocs             // Webルートディレクトリ
│   └── info.php       // PHP動作確認用
└── php               // Dockerfile、php.iniの格納ディレクトリ
    ├── Dockerfile     // PHPとApacheの融合コンテナを生成するための情報
    └── php.ini         // PHP設定ファイル
```



Visual StudioCodeなどのエディタを使ってこれらのファイルを作成します。

compose.yaml

YAML形式のファイルを作成し「docker compose」コマンドでファイルに記述されたコンテナのイメージ作成・起動を行います。(compose.yaml)

```
services:
  php-apache:
    build: ./php
    container_name: php-apache
    volumes:
      - ./htdocs:/var/www/html
    restart: always
    ports:
      - "8000:80"
    depends_on:
      - mysql
```

services:	生成・起動するサービス(コンテナ)を設定
php-apache:	サービス名(任意の名前)
build:	Dockerfileのあるディレクトリを指定
container_name:	コンテナ名(任意の名前)
volumes:	コンテナ内とホストOSのディレクトリを関連付ける Webルートのディレクトリを指定
restart:	コンテナが停止すると常に再起動する(always) 例えばコンテナ動作中にDockerが終了するとコンテナは 停止するが、Dockerを起動すると自動で再起動する
ports:	公開用のポート番号(ホスト:コンテナ)
depends_on:	コンテナの起動順序（指定したサービスの後に起動）

compose.yaml

```
mysql:
  image: mysql:latest
  container_name: mysql
  volumes:
    - db_data:/var/lib/mysql
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: 'password'
  ports:
    - "3306:3306"
```

mysql:	サービス名(任意の名前)
image:	DockerHubにあるイメージをそのまま利用するとき に指定
container_name:	コンテナ名(任意の名前)
volumes:	コンテナ内とvolumes:を関連付ける この設定でDBのデータを永続的に保存する
restart:	コンテナが停止すると常に再起動する(always) 例えばコンテナ動作中にDockerが終了するとコンテナは停止するが、Dockerを起動すると自動で再起動する
environment:	コンテナ内の環境変数を設定 MYSQL_ROOT_PASSWORD:はMySQLのrootパスワードを設定

compose.yaml

```
phpmyadmin:
  image: phpmyadmin:latest
  container_name: phpmyadmin
  depends_on:
    - mysql
  environment:
    PMA_HOST: mysql
    MEMORY_LIMIT: 128M
    UPLOAD_LIMIT: 64M
  restart: always
  ports:
    - "8080:80"
```

```
volumes:
  db_data: {}
```

phpmyadmin:	サービス名(任意の名前)
image:	DockerHubにあるイメージをそのまま利用するとき指定
container_name:	コンテナ名(任意の名前)
depends_on:	コンテナの起動順序（指定したサービスの後に起動）
environment:	PMA_HOST:利用するDBのサービス(mysql) MEMORY_LIMIT:ツールを利用するときの最大メモリ容量 UPLOAD_LIMIT:ファイルをアップロードするときの最大サイズ
restart:	コンテナが停止すると常に再起動する(always) 例えばコンテナ動作中にDockerが終了するとコンテナは停止するが、Dockerを起動すると自動で再起動する
ports:	公開用のポート番号(ホスト:コンテナ) phpmyadmin利用時のポート番号
volumes:	DBデータを永続化するための設定 db_dataの名前で保存される

Dockerfile

php-apacheイメージを生成・起動するためのDockerfileを作成します。
php-apacheはmod_phpを含んだApacheを生成・起動します。

- ホストOSで作成したphp.ini(PHP設定)を有効にする
- 画像/MySQL接続の処理を行うためのライブラリをインストールする

```
FROM php:8.0-apache
COPY ./php.ini /usr/local/etc/php/
RUN apt-get update &&\
    #PHP GD EXIF
```

PNG,JPEG,WEBP画像を扱うためのライブラリをインストール

```
    apt-get install -y zlib1g-dev libpng-dev libwebp-dev libjpeg62-turbo-dev libonig-dev &&\
    docker-php-ext-configure gd --with-jpeg --with-webp &&\
    docker-php-ext-install gd exif &&\
    #PHP PDO MySQL
    docker-php-ext-install pdo_mysql mysqli
```

PDOでMySQLと接続するための拡張をインストール

php.ini

PHPの設定ファイル

- 日本のタイムゾーン
- 日本語設定

```
[Date]
```

```
date.timezone = "Asia/Tokyo"
```

```
[mbstring]
```

```
mbstring.language = "Japanese"
```

サーバーの生成・起動

これで一通り必要な設定ファイルなどが完成したのでDocker Composeを使ってサーバーを生成・起動します。

Docker Desktopを起動し、次のコマンドをdocker-lamp配下で実行します。

【生成・起動】

```
docker compose up -d
```

このコマンド実行でディレクトリにあるcompose.yamlファイルに基づいてサービスが生成・起動されます。はじめての実行時にはイメージが生成されます。

【終了】

```
docker compose down
```

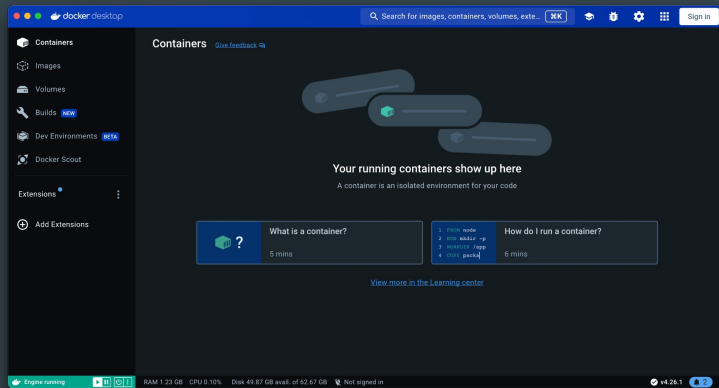
このコマンドで起動しているサービスを停止します。この場合イメージは残り、DBは保存されます。

【終了（イメージとDBの削除）】

```
docker compose down --rmi all --volumes
```

このコマンドでサービスを停止すると共に、イメージとDBも削除されます。

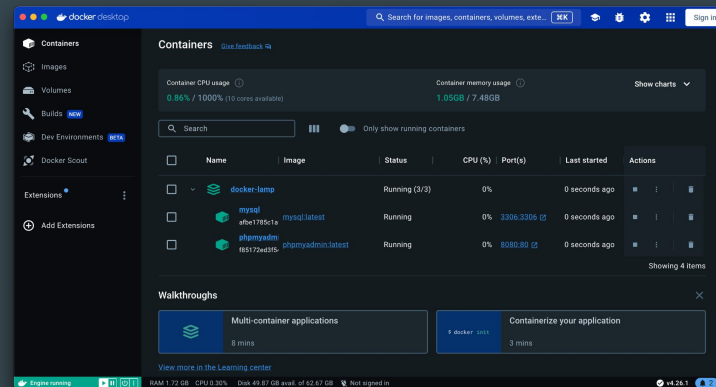
サーバーの生成・起動



コンテナが起動していない状態



コンテナが起動した状態



動作確認

サーバー環境の生成・起動ができた後、動作確認のため次のコードのphpファイルを作成しブラウザで「localhost:8000/info.php」にアクセスします。



info.php

```
<?php
phpinfo();
?>
```

[illegible]

ブラウザにPHPの
情報が表示され
ば動作している

※ロケールが日本になっていることを確認します。

Webアプリサンプル作成

次のコードはWebアプリのサンプルです。
(htdocs/sample.php)

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width,
initial-scale=1.0">
  <link href="css/style.css"
rel="stylesheet">
  <title>サンプル</title>
</head>
<body>
  <h1>サンプル</h1>
```

Dockerコンテナ間で接続する場合、ホスト名がサービス名orコンテナ名となります。

```
<?php
$now = new DateTime();
print $now->format('Y年m月d日G時i分s秒');

$dsn = 'mysql:host=mysql;dbname=sampladb;charset=utf8';
$user = 'root';
$password = 'password';

try {
  $db = new PDO($dsn, $user, $password);
  print '<p>接続成功</p>';

  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $db->exec("CREATE TABLE IF NOT EXISTS users(
    id INTEGER PRIMARY KEY,
    name VARCHAR(20),
    score INTEGER)");
  print '<p>テーブル作成</p>';
```

Webアプリサンプル作成

sample.phpのつづき

```
$db->exec("INSERT INTO users VALUES(1, 'Yamada', 85)");
$db->exec("INSERT INTO users VALUES(2, 'Tanaka', 79)");
$db->exec("INSERT INTO users VALUES(3, 'Suzuki', 63)");
print '<p>データ挿入</p>';

$q = $db->query("SELECT * FROM users WHERE score >= 70");
print '<p>70点以上選択</p>';
print "<p>";
while ($row = $q->fetch()) {
    print $row["id"] . " " . $row["name"] . " " . $row["score"] . "<br>";
}
print "</p>";
$db->exec("DROP TABLE users");
print '<p>テーブル削除</p>';

} catch (PDOException $e) {
    die ('エラー: ' . $e->getMessage());
}

?>
</body>
</html>
```

このサンプルでは次の処理をします。

- テーブル作成
- テーブルにデータ挿入
- テーブルからデータを抽出
- テーブル削除

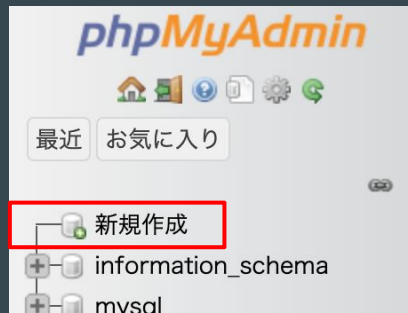
データベース作成

phpMyAdminを使ってデータベースを作成します。Webブラウザで「localhost:8080」に接続するとphpMyAdminのログイン画面が表示されます。



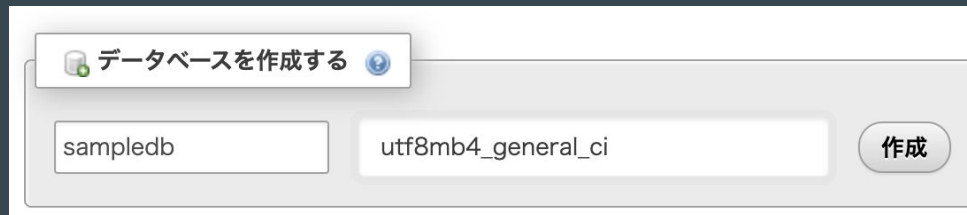
The login screen of phpMyAdmin. At the top is the phpMyAdmin logo and the text "phpMyAdmin へようこそ". Below this is a language selection dropdown menu set to "日本語 - Japanese". Underneath is a login section with fields for "ユーザ名:" (Username) and "パスワード:" (Password), and a "ログイン" (Login) button.

ユーザ名：root
パスワード：password



The main dashboard of phpMyAdmin. It features the phpMyAdmin logo and several navigation icons. Below the icons are tabs for "最近" (Recent) and "お気に入り" (Favorites). A red box highlights the "新規作成" (New) button. Below this, there are links for "information_schema" and "mysql".

データベース名：sampedb
照合順序
：utf8mb4_general_ciで作成



The screen for creating a new database. It has a title "データベースを作成する" (Create Database). Below the title are two input fields: the first contains "sampedb" and the second contains "utf8mb4_general_ci". To the right of these fields is a "作成" (Create) button.

style.css

スタイルシート(css/style.css)

```
h1 {  
  color: red;  
}
```

htdocs
├ css
│ └ style.css

Webルートであるhtdocs配下で任意の場所に配置します。
HTMLファイルやJavaScriptファイルなどの配置もhtdocs
配下で任意の場所となります。

動作確認

Webブラウザで「localhost:8000/sample.php」にアクセスしサンプルコードで作成したページが表示されることを確認します

PHPはコンテナを再起動せずともソースコード修正後は、ブラウザをリロードすると最新の状態で表示します。

※現在日時が日本の日時になっていることを確認しましょう。

サンプル

2022年12月30日19時13分16秒

接続成功

テーブル作成

データ挿入

70点以上選択

1 Yamada 85

2 Tanaka 79

テーブル削除

MySQLとSQLiteのどちらも同じように動作します。違いがわかるようにするには見出しなどを修正しましょう。

SQLite使用方法

PHPでSQLiteを使用するには、PDOオブジェクトを作るときにSQLiteでデータベースファイルを指定します。

```
$db = new PDO("sqlite:../db/sample.db");
```

これは、コードが動作するカレントディレクトリ配下のdbディレクトリにデータベースとなるsample.dbファイルを使用します。

SQLiteはファイルがなければ作成します（ディレクトリは存在しないとエラーになる）。SQL文はMySQLと基本的に同じものが使えますが、部分的に方言が存在する箇所もあります。先程のサンプルはSQL文はそのまま使えるので、PDOのインスタンス生成箇所だけを修正すればSQLiteを使用できるようになります。

Webサイト脆弱性の対策

Webサイトの脆弱性

Webサイトに関する脆弱性は様々な種類がありますが、ここでは課題に取り組むにあたり次の脆弱性について説明します。

- SQLインジェクション
- クロスサイト・スクリプティング

この資料ではこの2つの脅威についての基本情報と一般的な対策について説明します。詳細については以下のサイトを参照してください。

- 参照：<https://www.ipa.go.jp/security/vuln/websecurity.html>

SQLインジェクション

Webアプリの多くはデータベースと連携して動作しています。利用者からの入力情報を基にSQL文を組み立てていることも多く、そのSQL文に問題があった場合はデータベースの不正を招く可能性もあります。

このような問題をSQLインジェクションの脆弱性、不正を招くための攻撃をSQLインジェクション攻撃と言われています。

この攻撃により発生する脅威には「データベースの情報漏えい」「データベースの情報改ざん・消去」などがあります。

SQLインジェクションの動作例

次のコードは、先程作成したデータベースsampledbにフォームから入力した情報を保存・表示できるようにします。(threat1.php)

```
<?php
$dsn = 'mysql:host=mysql;dbname=sampledb;charset=utf8';
$user = 'root';
$password = 'password';
try {
    $db = new PDO($dsn, $user, $password);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->exec("CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(256))");

    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $name = $_POST['name'];
        if (isset($_POST['save'])) {
            $db->exec("INSERT INTO users (name) VALUES('$name')");
```

脅威動作例

名前:

番号:

usersテーブル作成
idカラムは番号を自動生成

「保存」ボタンを押してPOST
送信された名前をDBに保存

SQLインジェクションの動作例

threat1.phpのつづき

```
    } else if (isset($_POST['show'])) {  
        $id = $_POST['num'];  
        $q = $db->query("SELECT * FROM users WHERE id = $id");  
        $rows = $q->fetchAll();  
    }  
}  
} catch (PDOException $e) {  
    die ('エラー: '.$e->getMessage());  
}  
?  
<!DOCTYPE html>  
<html lang="ja">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>脅威動作例</title>  
</head>
```

「表示」ボタンを押してPOST
送信された番号のデータをDB
から取得

SQLインジェクションの動作例

```
<body>
  <h1>脅威動作例</h1>
  <?php if (isset($rows)): ?>
    <?php foreach ($rows as $row): ?>
      <p>
        「<?= $row['name'] ?>」さん、こんにちは！
      </p>
    <?php endforeach; ?>
    <a href="">戻る</a>
  <?php else: ?>
    <form method="POST">
      <p><label>名前:<input type="text" name="name"></label></p>
      <p><button type="submit" name="save">保存</button></p>
      <p>-----</p>
      <p><label>番号:<input type="text" name="num"></label></p>
      <p><button type="submit" name="show">表示</button></p>
    </form>
  <?php endif; ?>
</body>
</html>
```

threat1.phpのつづき

DBから取得したデータを画面
に表示

脅威動作例

名前:

保存

番号:

表示

SQLインジェクションの動作例

このコード(localhost:8000/threat1.php)にアクセスし、名前を入力し「保存」ボタンでDBに保存できることを確認します。複数保存します。(①)

保存された番号を入力し「表示」ボタンで画面にその名前が表示されることを確認します。(②)

① 脅威動作例

名前:

番号:

テーブル	操作
<input type="checkbox"/> users	★ 表示 構造 検索
1 テーブル	合計
↑ <input type="checkbox"/> すべてチェックする <input type="button" value="チェックする"/>	

				id	name
<input type="checkbox"/>	編集	コピー	削除	1	test
<input type="checkbox"/>	編集	コピー	削除	2	hoge
<input type="checkbox"/>	編集	コピー	削除	3	foobar

phpMyAdmin画面

② 脅威動作例

名前:

番号:

脅威動作例

「test」さん、こんにちは！

[戻る](#)

SQLインジェクションの動作例

番号の入力フィールドに「0 OR TRUE;」を入力し、「表示」ボタンを押すと保存しているすべての名前が画面に表示されることを確認します。この時点ではDBにデータは存在しています。

脅威動作例

名前:

保存

番号:

表示

脅威動作例

「test」さん、こんにちは！

「hoge」さん、こんにちは！

「foobar」さん、こんにちは！

[戻る](#)

SQL文は「SELECT * FROM users WHERE id = 0 OR TRUE;」となり、TRUEが有効なので全てのレコードが取得されます。これでフィールドに入力したSQL文が実行されることがわかります。

← T →					id	name
<input type="checkbox"/>	 編集	 コピー	 削除		1	test
<input type="checkbox"/>	 編集	 コピー	 削除		2	hoge
<input type="checkbox"/>	 編集	 コピー	 削除		3	foobar

phpMyAdmin画面

SQLインジェクションの動作例

番号の入力フィールドに「0; DELETE FROM users;」を入力し、「表示」ボタンを押すと保存している全てのレコードが削除されていることを確認します。

SQL文は「SELECT * FROM users WHERE id = 0; DELETE FROM users;」となり、DELETEが有効なので全てのレコードが消去されます。

脅威動作例

名前:

番号:

☐ プロファイリング [[イン](#)]

id	name
----	------

phpMyAdmin画面

SQLインジェクション対策

Webアプリの開発で、SQL文を発行する処理を実装する際は、プレースホルダーを用いてSQL文を組み立てる方法を適用します。

プレースホルダーは、SQL文の中に変数の場所を示す記号(プレースホルダー)を置いて、そこに実際の値を割り当てる処理を実行します。

データベースのエンジン側で値を割り当てることを静的プレースホルダーといい、アプリケーション側で値をエスケープして割り当てることを動的プレースホルダーと言います。どちらも有効な対策ですが、仕組みとして静的プレースホルダーの方が優位とされています。

静的プレースホルダーを用いて作成した文をPrepared Statementと言います。

SQLインジェクション対策

先程のコードを次のようにプレースホルダーを用いた処理に修正します。
(threat2.php)

```
try {
    $db = new PDO($dsn, $user, $password);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    --省略--
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $name = $_POST['name'];
        if (isset($_POST['save'])) {
            $stmt = $db->prepare("INSERT INTO users (name) VALUES(:name)");
            $stmt->bindParam(':name', $name, PDO::PARAM_STR);
            $stmt->execute();
        } else if (isset($_POST['show'])) {
            $id = $_POST['num'];
            $stmt = $db->prepare("SELECT * FROM users WHERE id = :id");
            $stmt->bindParam(':id', $id, PDO::PARAM_INT);
            $stmt->execute();
            $rows = $stmt->fetchAll();
        }
        --省略--
    }
}
```

静的プレースホルダーを有効

プレースホルダーを使って値を設定してSQL文を実行

SQLインジェクション対策

修正したコードthreat2.phpにアクセス(localhost:8000/threat2.php)し、番号の入力フィールドに「0; DELETE FROM users;」を入力し、「表示」ボタンを押しても保存しているレコードは削除されないことを確認します。

「SELECT * FROM users WHERE id = 0; DELETE FROM users;」のSQL文が実行されても、レコードは消去されない。

脅威動作例

名前:

番号:

脅威動作例

名前:

番号:

脅威動作例

名前:

番号:



名前を保存後
SQL文を入力

脅威動作例

名前:

番号:

↔ T ↔					▼	id	name
<input type="checkbox"/>		編集		コピー		削除	4 test
<input type="checkbox"/>		編集		コピー		削除	5 hoge
<input type="checkbox"/>		編集		コピー		削除	6 foobar

phpMyAdmin画面

クロスサイト・スクリプティング

Webアプリの利用者からの入力情報や、URLの末尾に付加したパラメータ情報を処理してWebページに出力した際、その情報にスクリプトが動作する内容が埋め込まれている可能性があります。

このような問題をクロスサイト・スクリプティングの脆弱性、この動作を悪用した攻撃をクロスサイト・スクリプティング攻撃と言われています。

この攻撃により発生する脅威には「偽サイト表示による情報漏えい」「Cookieの情報漏えい」「意図しないCookie情報の保存」などがあります。

クロスサイト・スクリプティングの動作例

先程作成したコードthreat2.phpにアクセス(`localhost:8000/threat2.php`)し、名前の入力フィールドに「`<script>alert('XSS');</script>`」を入力し、「保存」ボタンを押してDBに保存されることを確認します。

脅威動作例

名前:

保存

番号:

表示

			id	name
<input type="checkbox"/>	編集	コピー	削除	4 test
<input type="checkbox"/>	編集	コピー	削除	5 hoge
<input type="checkbox"/>	編集	コピー	削除	6 foobar
<input type="checkbox"/>	編集	コピー	削除	7 <script>alert('XSS');</script>

脅威動作例

名前:

保存

番号:

表示

保存された番号を入力して「表示」ボタンを押すと、スクリプトが実行されてポップアップが表示されます。

localhost:8000 の内容
XSS

OK

クロスサイト・スクリプティング対策

取得した情報をWebページに出力したときにスクリプトが動作するのは、情報に含まれたHTMLテキストのタグが機能するためです。

基本的にはHTMLテキストのタグが機能しないように文字列をエスケープします。ただし、アプリやサービスによってHTMLが機能するよう取り扱う必要がある場合は、アプリによって対策を変える必要があります。

たとえば<script>タグだけ動作しないようにするなどです。

ここでは課題に取り組むにあたり、すべてのHTMLテキストをエスケープする方法を説明します。

クロスサイト・スクリプティング対策

先程のコードthreat2.phpを次のように入力データをHTMLエスケープするよう修正します。(threat3.php)

```
--省略--
```

```
<body>
```

```
<h1>脅威動作例</h1>
```

```
<?php if (isset($rows)): ?>
```

```
<?php foreach ($rows as $row): ?>
```

```
<p>
```

```
「<?= htmlspecialchars($row['name']) ?>」さんこんにちは。
```

```
</p>
```

```
<?php endforeach; ?>
```

```
<a href="">戻る</a>
```

```
<?php else: ?>
```

```
--省略--
```

クロスサイト・スクリプティング対策

修正したコードthreat3.phpにアクセス(localhost:8000/threat3.php)し、HTMLタグが含まれたデータを持つレコードの番号を入力し、「表示」ボタンを押して動作を確認します。

脅威動作例

名前:

保存

番号:

表示

				id	name
<input type="checkbox"/>	編集	コピー	削除	4	test
<input type="checkbox"/>	編集	コピー	削除	5	hoge
<input type="checkbox"/>	編集	コピー	削除	6	foobar
<input type="checkbox"/>	編集	コピー	削除	7	<script>alert("XSS");</script>

脅威動作例

「<script>alert("XSS");</script>」さん、こんにちは！

[戻る](#)

HTMLタグは機能せず、ポップアップは表示されずに文字列として画面に表示されます。

Cookieの使い方

Cookieの使い方

Cookie(クッキー)はHTTP通信の中でクライアントにデータを保存する仕組みです。また一般的にはセッション管理は仕組みとしてcookieを使うが、PHPではセッション管理で直接cookieを使わず、session機能を利用します。

まずはcookieの基本的な使い方を説明します。

サンプルとしてWebサイトへのアクセス数のカウントをcookieを使って実現します。cookieはHTTPヘッダーの一部なので文字列を保存します。なのでカウントし保存する文字列を数値にして数えていきます。

Cookieの使い方

アクセス数：1回

カウントクリア

.....

アクセス数：12回

カウントクリア

サイトにアクセスする度に数が1ずつ増えていく。
カウントクリアを押すと0回になり、アクセスする度にまた1から増えていく。
有効期限を20秒に設定し、20秒後にまた1から増えていくようにする。

Cookieの使い方

次のコードを作成しクッキーの動作を確認します。(cookie.php)

```
<?php
    if (isset($_COOKIE['count'])) {
        $count = (int)$_COOKIE['count'];
        $count += 1;
    } else {
        $count = 1;
    }
    if (isset($_POST['clear'])) {
        setcookie('count', $count, time());
        header('Location: cookie.php');
        exit();
    } else {
        setcookie('count', $count, time()+20);
    }
?>
```

クッキーが保存されていれば、読み込んでその数値を1増加する。

そうでなければ\$count変数に1を代入する。

もしカウントクリアのボタンが押されたら、期限切れを今の時間にすることでクッキーを削除し、再びサイトにアクセスする。

クッキーの有効期限を20秒にする。

Cookieの使い方

つづき

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Cookieサンプル</title>
</head>
<body>
  <h2>アクセス数 : <?= $count ?>回</h2>
  <form method="POST">
    <button type="submit" name="clear">カウントクリア</button>
  </form>
</body>
</html>
```

アクセス数：1回

カウントクリア

コードを作成したら
localhost:8000/cookie.php
にアクセスする

Sessionの使い方

Sessionの使い方

例えばショッピングサイトやブログ投稿など、ユーザーや管理者などのログインを必要とする機能をWebアプリが提供する場合は、ログイン中を識別するためのセッション管理を行います。

ここではサンプルとしてログインページを表示し、ユーザー名とパスワードを入力してセッションを確立します。

ユーザー名は任意、パスワードは「1234」を固定としログインできるようにします。

ログイン中は「ユーザー名 - ログイン中です。」を表示し、ログアウトするとトップページへリダイレクトします。

セッションの有効期限を30秒に設定します。

Sessionの使い方

ログイン

ユーザー名 :

パスワード :

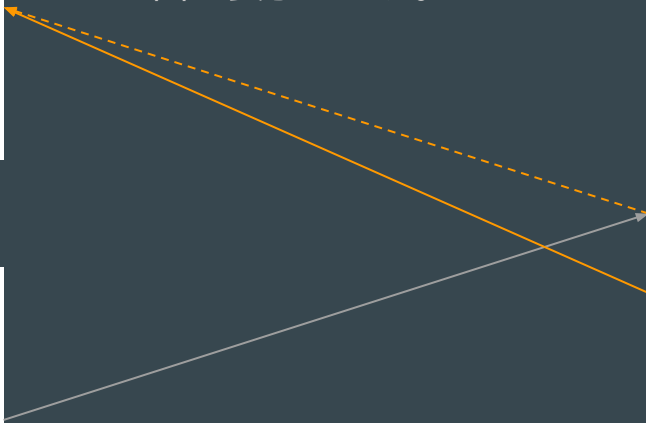
ログイン

ユーザー名 :

パスワード :

ログアウトを押すか、有効期限切れでセッションを破棄しトップページへリダイレクトし、ログイン画面を表示します。

hoge - ログイン中です。



Sessionの使い方

次のコードを作成しセッション管理の動作を確認します。(session.php)

```
<?php
$expire = 30;
ini_set('session.gc_divisor', 1);
ini_set('session.gc_maxlifetime', $expire);
session_set_cookie_params($expire);
session_start();

if (isset($_POST["logout"])) {
    $_SESSION = array();
    session_destroy();
    header('Location: session.php');
    exit();
}
if (isset($_POST['login'])) {
    if ($_POST['password'] != '1234') {
        print('<p>パスワードが違います</p>');
```

セッション有効期限30秒を設定。gc_maxlifetimeに有効期限を秒単位で設定するが、gc_divisorが初期値100のため、時間経過後も100/1の確立でセッションが破棄される。そのためこの値を1にする。

セッション有効期限は最後にセッションをはじめてからの時間となる。

有効期限が切れると\$_SESSIONに格納しているデータは破棄されるが、確実に有効期限をチェックし破棄するには\$_SESSIONへデータを格納する際、そのときの時間も格納し、もし\$_SESSIONが破棄されてなければ、現時間との差分をチェックし有効期限が過ぎていたら自身で破棄する処理を加えてもよい。

Sessionの使い方

session.phpのつづき

```
    } else {  
        $_SESSION['username'] = $_POST['username'];  
    }  
}  
?>
```

```
<!DOCTYPE html>  
<html lang="ja">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Sessionサンプル</title>
```

```
</head>
```

```
<body>
```

```
    <?php if (isset($_SESSION['username'])) { ?>
```

```
        <p><?php echo htmlspecialchars($_SESSION['username']); ?> - ログイン中です。</p>
```

```
        <form method="post">
```

\$_SESSIONへデータを格納することでセッションが確立する。ここではユーザー名を格納している。これに加え、このときの時間を\$_SESSIONへ格納し有効期限をチェックするのもよい。
例) \$_SESSION['last_time'] = time()

Sessionの使い方

```
<button type="submit" name="logout">ログアウト</button>
</form>
<?php } else { ?>
<h2>ログイン</h2>
<form method="post">
  <label>ユーザー名 : <input type="text" name="username" required></label><br>
  <label>パスワード : <input type="password" name="password" required></label><br>
  <button type="submit" name="login">ログイン</button>
</form>
<?php } ?>
</body>
</html>
```

session.phpのつづき

コードを作成したら「localhost:8000/session.php」
にアクセスする

ログイン

ユーザー名 :
パスワード :

パスワードの保存について

パスワードの保存について

Sessionの使い方ではログインを維持するコードを書きましたが、そこではパスワードを「1234」固定で実装しました。通常ログインは登録されている情報と照合して正しい場合にセッションを確立します。

ユーザー名とパスワードを記録し、ログイン時にそのユーザー名とパスワードを照合します。

そのパスワードを記録する際に平文ではなく、ハッシュ化したデータを記録することで、情報漏洩してもパスワードをわかりにくくします。

※ハッシュ化とは、一方向に暗号化するアルゴリズムで復号化できないためパスワードを保存する際に利用されている。

パスワードの保存について

サンプルとして1つの画面にログインと登録のフォームを表示し、ユーザー名とパスワードを入力し、ログインボタンを押したときはログインを試みます。

登録ボタンを押したときは、パスワードはハッシュ化してユーザー名とパスワードをJSONファイルに保存します。すでに登録されているユーザーは、パスワードを上書きします。

Sessionの使い方で作成したログインするコードを修正し、記録したユーザー名とパスワードで照合しログインします。

ログインが成功した場合は「ログイン中」を表示し、ログインできなかった場合は「ログイン失敗・パスワードが違います」「ログイン失敗・ユーザーが登録されていません」を表示します。Session有効期限は30秒とします。

パスワードの保存について

ログインフォームとユーザー登録フォームを表示します。ログイン中はセッションを確立します。ユーザー登録したユーザー名とパスワードはJSONファイルに保存されます。保存の際、パスワードをハッシュ化します。

パスワード保存とログイン

ログイン

ユーザー名:
パスワード:

ログイン

ユーザー登録

ユーザー名:
パスワード:

登録

パスワード保存とログイン

ログイン

ユーザー名: hoge
パスワード:

ログイン

ユーザー登録

ユーザー名:
パスワード:

登録

動作内容

パスワード保存とログイン

foobar - ユーザーが登録されていません。

[ログインへ](#)

エラー表示
登録されてない

ユーザー登録

ユーザー名:
パスワード:

登録

パスワード保存とログイン

hoge - パスワードが違います。

[ログインへ](#)

エラー表示
パスワードが違う

ユーザー登録

ユーザー名:
パスワード:

登録

パスワード保存とログイン

hoge - ログイン中です。

セッション確立

ログアウト

ログアウトすると
入力画面に戻る

ユーザー登録

ユーザー名:
パスワード:

登録

パスワードの保存について

htdocs/hash_sample.php

Webアプリのコード作成

```
<?php
    $expire = 30;
    ini_set('session.gc_divisor', 1);
    ini_set('session.gc_maxlifetime', $expire);
    session_set_cookie_params($expire);
    session_start();

    function hash_password($password) {
        $str = '1234567890abcdefghijklmnopqrstuvwxyz';
        $salt = substr(str_shuffle($str), 0,16);
        $hash = hash('sha256',$password.$salt);
        return $salt.$hash;
    }
```

セッションの有効期限を設定

ハッシュ化するhash_password関数を定義
ソルト値を加えsha256でハッシュ化したテキストを返却する

パスワードの保存について

hash_sample.phpのつづき

```
function verify_password($password, $hash) {  
    $salt = substr($hash, 0, 16);  
    $digest = substr($hash, 16);  
    $verify = hash('sha256', $password.$salt);  
    return $verify == $digest;  
}
```

```
if (isset($_POST['signup'])) {  
    if (file_exists("hash_sample.json")) {  
        $json = file_get_contents("hash_sample.json");  
        $json = mb_convert_encoding($json, "UTF-8");  
        $signupinfo = json_decode($json, true);  
        $signupinfo[$_POST['username']] = hash_password($_POST['password']);  
    }
```

パスワードが同一かをチェックする
verify_password関数を定義
入力したパスワードと保存している
ハッシュ化したパスワードを渡し
チェックする

登録ボタンを押したときにJSON
ファイルにある情報に追加で登録す
る

パスワードの保存について

hash_sample.phpのつづき

```
} else {  
    $signupinfo = array($_POST['username'] => hash_password($_POST['password']));  
}  
$signupinfo = json_encode($signupinfo);  
file_put_contents("hash_sample.json",$signupinfo);  
}  
  
if (isset($_POST["logout"])) {  
    $_SESSION = array();  
    session_destroy();  
    header('Location: hash_sample.php');  
    exit();  
}
```

ログアウトボタンを押すとセッション情報を削除しトップヘリダイレクトする

パスワードの保存について

hash_sample.phpのつづき

```
if (isset($_POST['login'])) {  
    if (file_exists("hash_sample.json")) {  
        $json = file_get_contents("hash_sample.json");  
        $json = mb_convert_encoding($json, "UTF-8");  
        $signupinfo = json_decode($json, true);  
        foreach ($signupinfo as $key => $value) {  
            if ($key == $_POST['username']) {  
                $username = $_POST['username'];  
                if (verify_password($_POST['password'], $value)) {  
                    $password = $value;  
                }  
            }  
        }  
    }  
}
```

ログインボタンを押したとき、もしJSONファイルが存在していれば、ファイルから取得したユーザー名とパスワードを確認する

パスワードの保存について

hash_sample.phpのつづき

```
if (isset($username) && !isset($password)) {  
    $error = 'パスワードが違います。';  
} else if (!isset($username) && !isset($password)) {  
    $username = $_POST['username'];  
    $error = 'ユーザーが登録されていません。';  
} else {  
    $_SESSION['username'] = $_POST['username'];  
}  
}
```

ユーザー名は一致したがパスワードが一致しないときは「パスワードが違います」、ユーザー名が一致しなかったときは「ユーザーが登録されていません」を表示する。
ユーザー名とパスワードが一致したときは、ユーザー名をセッションに設定する。

パスワードの保存について

hash_sample.phpのつづき

```
?>
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>パスワード保存とログイン</title>
</head>
<body>
  <h1>パスワード保存とログイン</h1>
  <?php if (isset($_SESSION['username'])) { ?>
    <p><?php echo htmlspecialchars($_SESSION['username']); ?> - ログイン中です。</p>
    <form method="POST">
      <button type="submit" name="logout">ログアウト</button>
    </form>
```

HTMLを記述する
エラーがあればエラーを表示する

パスワードの保存について

hash_sample.phpのつづき

```
<?php } else { ?>
    <?php if (isset($username) && isset($error)) { ?>
        <p><?php echo htmlspecialchars($username); ?> - <?php echo $error; ?></p>
        <?php $error = "" ?>
        <a href="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">ログインへ</a>
    <?php } else { ?>
        <h3>ログイン</h3>
        <form method="POST">
            <label>ユーザー名 : <input type="text" name="username" required></label><br>
            <label>パスワード : <input type="password" name="password" required></label><br>
            <button type="submit" name="login">ログイン</button>
        </form>
    <?php } ?>
<?php } ?>
```

ログインできればログイン中を表示する
エラーもなくログイン中でもないければログイン
フォームを表示する

パスワードの保存について

hash_sample.phpのつづき

```
<p>-----</p>
<h3>ユーザー登録</h3>
<form method="POST">
  <label>ユーザー名 : <input type="text" name="username" required></label><br>
  <label>パスワード : <input type="password" name="password" required></label><br>
  <button type="submit" name="signup">登録</button>
</form>
</body>
</html>
```

ユーザー登録フォームを表示する

パスワードの保存について

URL「localhost:8000/hash_sample.php」にアクセスし動作確認する。

パスワードをハッシュ化して保存する

[hash_sample.json]

```
{  
  "hoge":  
    "1c30fb408bfbd9c3745e13e9c5aad4b3eb01f56e0747ae  
    2b6728bdd842240eab9"  
}
```

パスワード保存とログイン

ログイン

ユーザー名:

パスワード:

ユーザー登録

ユーザー名:

パスワード:

パスワード保存とログイン

ログイン

ユーザー名:

パスワード:

ユーザー登録

ユーザー名:

パスワード:

登録

パスワード保存とログイン

ログイン

ユーザー名:

パスワード:

ログイン

パスワード保存とログイン

hoge - ログイン中です。

登録したユーザー名
とパスワードでログイン可能

パスワード保存とログイン

hoge - パスワードが違います。

[ログインへ](#)

エラー表示

パスワード保存とログイン

foobar - ユーザーが登録されていません。

[ログインへ](#)

パスワードの保存について

PHPは標準でパスワード用にハッシュ化の機能を提供しています。この機能を使うとさらに高度なセキュリティ対策されたハッシュ化を利用することができます。

以下の関数を使いハッシュ化したりパスワードを判定します。

- `password_hash()` #パスワードをハッシュ化する
- `password_verify()` #ハッシュ化したパスワードと元のパスワードを比較し、同一であるかを判定する

パスワードの保存について

--省略--

```
if (isset($_POST['signup'])) {  
    if (file_exists("hash_sample.json")) {  
        $json = file_get_contents("hash_sample.json");  
        $json = mb_convert_encoding($json, "UTF-8");  
        $signupinfo = json_decode($json, true);  
$signupinfo[$_POST['username']] = hash_password($_POST['password']);  
        $signupinfo[$_POST['username']] = password_hash($_POST['password'], PASSWORD_DEFAULT);  
    } else {  
$signupinfo = array($_POST['username'] => hash_password($_POST['password']));  
        $signupinfo = array($_POST['username'] => password_hash($_POST['password'], PASSWORD_DEFAULT));  
    }  
}
```

--省略--

パスワードの保存について

--省略--

```
if (isset($_POST['login'])) {  
    if (file_exists("hash_sample.json")) {  
        $json = file_get_contents("hash_sample.json");  
        $json = mb_convert_encoding($json, "UTF-8");  
        $signupinfo = json_decode($json, true);  
        foreach ($signupinfo as $key => $value) {  
            if ($key == $_POST['username']) {  
                $username = $_POST['username'];  
if (verify_password($_POST['password'], $value)) {  
                if (password_verify($_POST['password'], $value)) {  
                    $password = $value;  
                }  
            }  
        }  
    }  
}
```

--省略--