CHAPTER 20

インターネットに公開するときの対策

これまでのlocalhostで学んできた知識を使って、「さっそくWebアプリケーションをレンタルサーバーに置いて公開！」と考えた方もいるかもしれません。でも、ちょっと待ってください！

インターネットには、不特定多数の人が行き交っています。あらゆる事態に対して備えておかなければ、大事なデータベースが危険にさらされてしまいます。

インターネット上にWebアプリケーションを公開するには、それなりの対策が必要です。ここでは、Webアプリケーションを安全に運用するのに必要な、最小限の知識について触れます。

公開されるフォルダに重要な情報を置かない

PHPファイルの仕組み

Webページを公開するときは、Webサーバー上の公開されるフォルダにファイルを置きます（→P.344）。公開されたファイルは、インターネットに接続できる人ならば、もちろん誰でもダウンロードが可能になります。

では、データベースを操作するP.423のようなスクリプトも、接続した人にダウンロードされてしまうのでしょうか？　もし、誰でも自由にダウンロードができて、その内容を見ることができるのなら、サーバー名やユーザー名、そしてパスワードまでをも、世間に公開していることになります。

実は、公開されるフォルダでPHPが動作する設定になっていれば、「.php」の拡張子を持つファイルにアクセスするとスクリプトが実行されます。PHPスクリプトファイルにアクセスしたクライアントに返されるのは、スクリプトが実行した結果だけです。つまり、「.php」の拡張子を持つファイルの内容は、公開されることはありません。

さらには、サーバー上にあるファイルの内容を他人に見せたくない場合、一般的には属性を変更してアクセスできないように制限をかけます。

だから、重要な内容を含むファイルも「PHPスクリプトにして、アクセス制限をかければ絶対安心！」といいたいところなのですが。Webアプリケーションを構成するシステムにセキュリティ上の欠陥がある場合など、本来見られないはずのファイルが、どういうわけか見られてしまうこともあるのです。そして、インターネットを利用する人には、悪意のある人も存在するでしょう。ですから、パスワード等の重要な情報

が書き込まれたファイルを、公開するフォルダに置くことは大変危険なことだと思ってください。

では、重要な情報をスクリプトで扱うときは、どうしたらよいのでしょうか？

別ファイルのスクリプトを読み込む方法

Webサーバーとなるコンピュータには、必ず「公開されないフォルダ」が設定できます。重要な情報を含むファイルは、必ず「その場所が推測されにくい、非公開のフォルダ」に置きましょう。公開されるフォルダに置くファイルには重要な情報を記述せず、必要な情報は別の場所から読み込むようにすればよいのです。

require_once

PHPスクリプトには、別のPHPスクリプトファイルを読み込む機能がいくつか用意されています。ここでは例として、require_once命令を使うことにします。この命令でファイルを読み込んだ場合、「ファイルを一度だけ」読み込み、「読み込めない場合は処理を中止」します。

書式 require_once

```
require_once(読み込みファイル名)
```

まず、「サーバー名」「ユーザー名」「パスワード」「データベース名」を記述したファイルを作ります。このファイルは公開されないフォルダに保存します。

List20-01のようなファイルを作成し、ファイル名を「db_info.php」として、公開されないフォルダ「data」に保存するものとします。

LIST 20-01 db_info.php

```
<?php
$SERV="localhcst";
$USER="root";
$PASS="root";
$DBNM="db1";
?>
```

P.446のデータベースに接続するスクリプト「kantan_select.php」を改良して、この基本情報を保存したファイル「db_info.php」を読み込むようにします。

変更するのは、データベースに接続する次の部分だけです。

・・・▶ 変更前

```
$s=new PDO("mysql:host=localhost;dbname=db1","root","root");
```

・・・▶ 変更後

```
require_once("data/db_info.php");
$s=new PDO("mysql:host=$SERV;dbname=$DBNM",$USER,$PASS);
```

「require_once ("data/db_info.php")」では、dataフォルダにある「db_info.php」を読み込んでいます。

ここではわかりやすくするために、仮に公開されるフォルダ(htdocsなど)の中に単純にdataフォルダを作成し、そこに「db_info.php」を置く形にしています。しかしもちろん、本来であればパスワード等を記述したファイルを置くフォルダは、公開されるフォルダより上位の、公開されない安全な場所に置く必要があります。

変更後の全体のスクリプトは、List20-02のようになります。

LIST 20-02 kantan_select2.php

```
<?php
require_once("data/db_info.php");
$s=new PDO("mysql:host=$SERV;dbname=$DBNM",$USER,$PASS);

$re=$s->query("SELECT * FROM tbk ORDER BY bang");
while($kekka=$re->fetch()){
    print $kekka[0];
    print " : ";
    print $kekka[1];
    print " : ";
    print $kekka[2];
    print "<br>";
}

print "<br><a href='kantan.html'>トップメニューに戻ります</a>";
?>
```

単純に、「require_once ("data/db_info.php");」の部分に読み込んだファイルの内容が挿入されると考えてよいでしょう。

同様の変更を他の「kantan_insert.php」「kantan_delete.php」「kantan_search.php」にも加えれば、もしスクリプトの内容を見られたとしても、パスワード等まで見られてしまうことはなくなります。

COLUMN ▶ **外部ファイルを取り込むコマンド**

「require_once」以外にも、外部ファイルを取り込む命令がいくつか用意されています。たとえば「include_once」命令は、ファイルを一度だけ読み込み、読み込めない場合も処理を続けます。

クエリに不正なデータを入れられないようにする

SQLインジェクションとは

パスワード等、重要な情報が見られなければPHP+MySQLは安心なのか、というと、決してそんなことはありません。むしろ、もっと怖くてやっかいなのが**SQLインジェクション**です。

インジェクション(injection)は、「注射する」とかお金を「つぎ込む」といった意味に使われます。つまり「SQLインジェクション」は、「SQL文を"注入"する」というような意味になります。「Webアプリケーションの作者が予想していなかったデータ」を混ぜる(注入する)ことで、不正な処理を行わせる攻撃方法です。

では、SQLインジェクションの恐怖を体験してみましょう。

P.450のPHPスクリプト「kantan_delete.php」を思い出してください。第19章で作った、簡単掲示板ですね。この中に「指定した番号のレコードを削除する」という機能がありました。次が、それに該当する部分です。

```
$b1_d=$_POST["b1"];
$s->query("DELETE FROM tbk WHERE bang=$b1_d");
$re=$s->query("SELECT * FROM tbk ORDER BY bang");
```

送信側である「kantan.html」では、削除するレコードの番号を「<input … name="b1">」のテキストボックスに入力してsubmitします。

すると、受け取り側の「kantan_delete.php」が「$_POST ["b1"]」で番号を受け取り、「$b1_d=$_POST ["b1"]」で「$b1_d」に代入し、これを元に次のようなクエリを発行します。

```
DELETE FROM tbk WHERE bang=$b1_d
```

私たち作成者としては、「$b1_d」には当然、存在するレコードの番号の「6」とか「12」が送られることを予想しています。

SQLインジェクションを体験する

ここでちょっと考えてみてください。「kantan.html」の、削除番号を入力するテキ

ストボックスに、次のように入力して［送信］ボタンをクリックしたらどうなるでしょうか？

```
1 OR 1=1
```

最初の「1」で、1番目のレコードが削除されそうですね。でも、その次の「OR 1=1」というのは、何ものなのでしょうか？

これを上のSQL文に入れてみると、次のようになります。

```
DELETE FROM tbk WHERE bang=1 OR 1=1
```

ここで気が付いた方もいることでしょう。そうです、「1=1」というのは絶対に正しいのです（1と1は同じ！）。ですから、「bangが1のとき」または「1=1のとき」は、「すべてが正しく」なってしまうのです。

これはつまり、1番目のレコードが削除されるだけでなく、テーブル「tbk」のすべてのレコードが削除されてしまうことになります。

FIG 20-01 1 OR 1=1と入力すると…

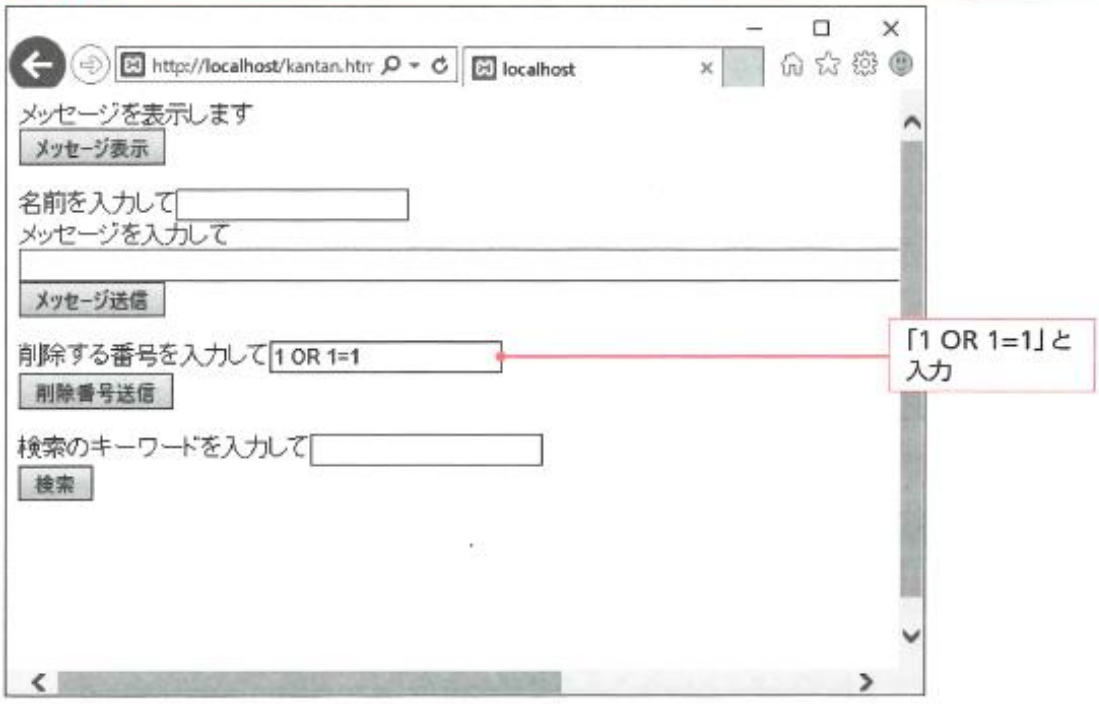


FIG 20-02 実行結果

実際にこのサンプルで「1 OR 1=1」を送信すると、すべてのレコードが削除されてしまいます。このように簡単に、「Webアプリケーションの作者が意図しない、危険なクエリ」を発行することができてしまうのです。

このように、SQL文の一部を含めてデータを送信することで、システムを攻撃する方法が「SQLインジェクション」です。

もっとも、第19章のサンプル「kantan_delete.php」は、もともと「どうぞ御自由に削除してください」という仕組みのスクリプトです。1レコードずつ削除しても、一度に全部削除しても同じことなので実害はありません。しかし、これがもしインターネットに公開し、本格的に運用しているデータベースの重要部分だったらどうでしょうか。商取引などで大事な顧客のデータを紛失してしまったら、目も当てられません。

世の中には、もっともっと巧妙な「SQLインジェクション」を使って、さまざまな攻撃を仕掛けてくる人がいます。たとえスクリプトの中身や変数が全部見えなくても、変数や処理の流れを推測して危険なSQL文の破片を注入してくるのです。

それではWebアプリケーションを公開するとき、いったいどのような対策をとればよいのでしょうか？

数値以外を入力させなくする

たとえば上の例なら、「数値以外のデータが送信されたら、DELETEを実行させない」という仕組みを作ればよいでしょう。送信されるデータは、削除するレコード番号のはずなので、もしアルファベットが含まれている場合は削除せず、警告メッセージを送るようにするのです。

「アルファベットが含まれている」ことをチェックする方法は、いろいろありますが、ここでは「正規表現」と「preg_match」関数を使った方法を解説します。

正規表現

正規表現とは

正規表現とは、「このような文字のパターン（秩序やルール）になっているかどうか」という条件を記述する方法です。たとえば、「[0-9]」という表現は、「0から9までの数字が含まれている」ということを意味します。

正規表現の例

ここでは、以下に代表的な例を紹介します。

[ ]の中の文字がどこかに含まれている

正規表現	内容
[7]	どこかに7が含まれている
[0-9]	どこかに数字が含まれている
[a-z]	どこかに小文字のアルファベットが含まれている
[A-Z]	どこかに大文字のアルファベットが含まれている
[A-Za-z]	どこかに大文字かまたは小文字のアルファベットが含まれている
[A-Z][0-9]	どこかに、最初が「大文字のアルファベット」で次が「数字」、という連続した文字のパターンが含まれている

[ ]の中で指定した文字以外が含まれている

正規表現	内容
[^0-9]	0から9の文字以外が含まれている（数字以外が含まれている）
[^A]	A以外が含まれている
[^A-Z]	大文字のアルファベット以外が含まれている
[^0-9a-zA-Z]	数字とアルファベット以外が含まれている

^の次の文字で始まっている

正規表現	内容
^h	hで始まっている

$の前の文字で終わっている

正規表現	内容
E$	Eで終わっている

{ }の前の文字が{ }内の回数だけ連続している

正規表現	内容
7 {3}	どこかに7が3個以上連続して含まれている

preg_match関数

正規表現を使ってあいまい検索を行う関数に、preg_matchがあります。「あいまい

検索」とは、「この文字を探しなさい」というような厳密な指定ではなく、「だいたいこんな感じの文字を探しなさい」というような、対象範囲を広めに取った検索です。

preg_match関数は、次のように記述します。

書式 preg_match関数

```
preg_match(正規表現,調査する文字列)
```

「preg_match」関数では、非常にポピュラーなスクリプト言語であるPerlと互換性のある正規表現を使って検索を行います。

この場合、正規表現の記述を一般に「/」(スラッシュ)で囲みます。たとえば、「どこかに大文字かまたは小文字のアルファベットを含む」という正規表現は、[A-Za-z]です。次は、「AからZ、そしてaからz、つまりアルファベットが含まれている」という正規表現です。

```
/[A-Za-z]/
```

List20-03は「1234」を、「[/ A-Za-z] /」という正規表現でチェックするものです。「1234」はアルファベットを含まないため、「含みません」を表示します。

LIST 20-03 fukumu.php

```
<?php
if(preg_match("/[A-Za-z]/","1234")){
        print "含みます";
}else{
        print "含みません";
}
```

正規表現を使って不正入力をチェックする

もう1つ、正規表現の例を紹介します。「×××-××××」の形式の郵便番号が入力されていることをチェックしてみましょう。

- 「0～9が3つで始まる」 → 「-」 → 「0～9が4つで終わる」

という文字の並びが含まれているか？　という表現は、次のようになります。

- 「0～9が3つ」で　→　[0-9]{3}
- 「始まる」　→　^

- 「0～9が4つ」で　→　[0-9]{4}
- 「終わる」　→　($)

List20-04は、変数「$m」の内容「107-0052」に対して、上のチェックを行う例です。

LIST 20-04 seiki.php

```
<?php
$m="107-0052";
if(preg_match("/^[0-9]{3}-[0-9]{4}$/",$m)){
    print "とりあえずOKです";
}else{
    print "誤りがあります";
}
?>
```

この場合、正しいので「とりあえずOKです」と表示されます。

もちろん、実際にインターネットで公開するためには、これだけでは不十分です。厳密に調べるためには、いろいろな正規表現を加えていく必要があります。

数値以外ならクエリを実行しないように改良する

では、とりあえずP.450で紹介した「$s->query ("DELETE FROM tbk WHERE bang=$b1_d") ;」の部分で、数値以外の入力があった場合は警告を表示し、DELETE命令を実行しないように書き換えてみることにしましょう。

「$s->query ("DELETE FROM tbk WHERE bang=$b1_d") ;」の部分を、$b1_dの内容に数値以外が含まれる場合は次を書き出すことで警告を表示し、DELETE命令が実行されないように書き換えましょう。

数値以外が含まれる場合に書き出すコマンドは、次のようになります。

```
print "<div style='color:red'>数字以外は入力しないで！！</div>";
```

List20-05は、改良したkantan_delete.phpの削除実行部分です。

LIST 20-05 kantan_delete.phpの一部

```
if(preg_match("/[^0-9]/",$b1_d)){
    print "<div style='color:red'>数字以外は入力しないで　！</div>";
}else{
    $s->query("DELETE FROM tbk WHERE bang=$b1_d");
}
```

変数「$b1_d」の値に、もしアルファベットなどが入力(0～9以外が入力)されると、「数字以外は…」のメッセージが表示され、DELETE命令を含むクエリが実行されることはなくなります。「数字以外は…」の「 ～ ";」が設定された文字列は赤色になります(→P.400)。

そしてアルファベットなどが入力(0～9以外が入力)されなければ、「$s->query("DELETE FROM tbk WHERE bang=$b1_d");」で削除が実行されます。

もちろん万全を期すなら、「数値以外か？」だけでなく、適切な番号が入力されているかを厳密にチェックしたいところです。

昨今のSQLインジェクションにはさまざまな手口があり、そのすべてを解説することはできません。でも、ひとまずはこれだけでも、余計なSQL文の混入は防げるというレベルの仕組みになります。

意図しないタグを実行させない

悪意のあるタグの送信

タグの取り扱いも、Webアプリケーション作成時の重要なポイントです。

また、P.409の「okuri.html」「uke.php」で実験してみましょう。この2つのファイルは、「okuri.html」のテキストボックスに文字を入力して[送信]ボタンをクリックすると、「uke.php」が単純にこれをprintで書き出すという仕組みでした。

では、「uke.php」が動作するようにしておいてください。そしてlocalhostにある「okuri.html」を表示します。テキストボックスには次のように「<body bgcolor=black>」と入力して、[送信]ボタンをクリックしてみてください。

```
<body bgcolor=black>
```

結果はどのようになりましたか？

FIG 20-03 「okuri.html」を実行

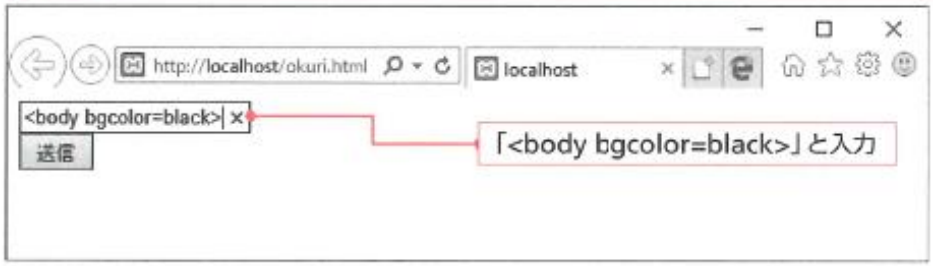
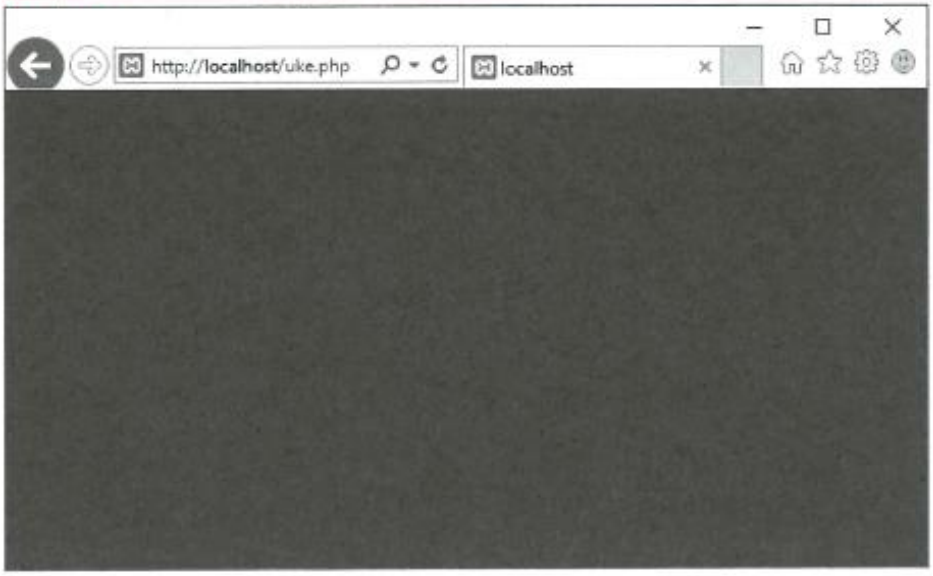


FIG 20-04 実行結果(「uke.php」)



いきなりウィンドウの中が真っ黒になってしまいました！　いったい何が起こったのでしょうか？

「okuri.html」では、単純にテキストボックスに入力された文字を「uke.php」に送るだけ、そして「uke.php」は受け取った文字列をprintで書き出すだけです。つまり、真っ黒になったブラウザの画面には「<body bgcolor=black>」が書き出されていることになります。<body>タグのbgcolor属性は、HTML5では使えなくなってしまった属性ですが、たいていのブラウザでは動作してしまいます。

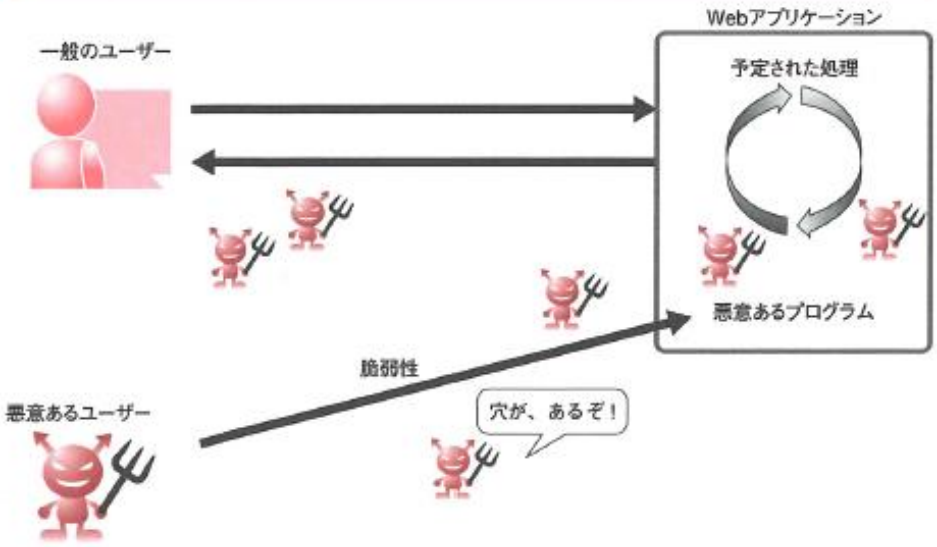
「okuri.html」「uke.php」のようなWebアプリケーションでは、入力されたタグにより、その設定された機能が働いてしまうのです。この程度のイタズラならかわいいものです。しかし、スクリプトを実行するタグのように、場合によっては大きな被害が出るものが送りつけられることも考えられます。インターネットに公開するには、万全の対策が必要です。

脆弱性への攻撃

上記のように、Webページでの入力をそのまま画面に書き出すプログラムは危険です。悪意ある人から回り回って送られたスクリプトによって、ユーザーが悪意あるスクリプトを意図しなくても実行してしまう。このようなシステムの脆弱性をねらった攻撃が後を絶ちません。

こうしたトラブルを防ぐためにも、入力されたスクリプトを排除するWebアプリケーションを作る必要があります。

FIG 20-05 脆弱性への攻撃



入力されたタグを取り除く

入力された文字を、そのままWebページに書き出す仕組みは危険です。とりあえず、書き出す文字列に含まれるタグを無効にする仕組みを設定しておきましょう。

このような場合には、タグなどの特殊文字を別の文字列に変換する「htmlspecialchars」関数を使います。

書式 htmlspecialchars関数

```
htmlspecialchars(文字列)
```

htmlspecialchars関数は、タグなどの特殊文字列を次のように変換します。

TABLE 20-01 htmlspecialchars関数による変換

変換対象文字	変換後
<	<
>	>
&	&
"	"
'	'

※ただし「'」(シングルクオート)の変換は、第2引数に「ENT_QUOTES」を指定した場合に行われる。

悪意ある危険な入力でよく使われるのは、「<」「>」「&」「"」「'」などの記号です。htmlspecialchars関数は、これらの文字列を、その機能が実行できない文字列に変換します。

たとえばP.411の「uke.php」に、次のようにhtmlspecialcharsを使ってみましょう。

LIST 20-06 uke_anzen.php（抜粋）

```
<?php
print htmlspecialchars($_POST["a"]);
?>
```

単純に、受け取った値が代入されている変数、「$_POST ["a"]」に対してhtmlspecial charsによる処理を加えただけです。

このようにスクリプトを変更したら保存して、再び「okuri.html」を実行します。テキストボックスに再び「<body bgcolor=black>」と入力して［送信］ボタンをクリックするとどうなるでしょうか？

FIG 20-06 実行結果

今度は、入力したタグがそのまま表示されました。つまり、タグとしての機能を奪うことができた、ということです。

次は表示された実行結果のソースを表示したものです。「<」と「>」が、それぞれ「<」と「>」に変換されているのがわかります。

FIG 20-07 ソース表示

COLUMN ▶ **安全なスクリプトを作成するには**

私たちはWebアプリケーションに対する攻撃に、可能な限りの対策を取っていく必要があります。では具体的には、いったいどのような対策が必要なのでしょうか？

その主だったものを挙げます。

・公開されるフォルダに置くファイルは、最小限にする（→P.457）。パスワード等の情報やデータベースそのものなど、重要なデータは決して置かない。ファイルやフォルダにはアクセスの制限をかける
・入力されたデータを厳密にチェックし、指定された形式以外の値を無効にする仕組みを作る（→P.462）
・送信された値によって、作成者が意図しない動作が起こらない構造にする

以上の対策を行えば安心なのかというと、そんなことはありません。Webアプリケーションに対する「攻撃手法」は、日々進化しているのです。

普段利用しているHTTPプロトコル（→P.336）は、基本的には1回の「要求→応答」で終わるものです。このような、個人認証なしで誰でもが利用できるシステムでは正直なところ、「攻撃に対する万全の対策」を設定するのは困難だと言えるでしょう。

私たちは、最悪の事態を想定し、そして常に可能な限りの対策を考えていく必要があるのです。

まとめ

この章では、以下のような事柄について解説しました。

・インターネットにWebアプリケーションを公開するときに注意すべき点
・パスワード等重要な情報を含むファイルの扱い方
・SQLインジェクションへの対策の例
・タグを混入させる攻撃への対策の例

セキュリティ対策は、決して怠ることがあってはいけません。それは、自分の犯したミスが、自分だけの問題ではなく、不特定多数の人たちに迷惑を掛けてしまう可能性があるからです。無論、この章で勉強したことだけで十分、というわけではありません。常に、セキュリティ対策を意識することが重要です。

▶ チェックしてみよう

この章で学んだ以下の事柄を理解しているかチェックしてみましょう。

☐ SQLインジェクションとは何であるかを理解している

- ❒ require_once関数を使って、別のファイルのスクリプトを読み込むことができる
- ❒ タグを無効にする方法を理解している
- ❒ preg_match関数と正規表現を使ったチェックの方法がわかる

練習問題

問題1

JavaScriptの命令は、次の<script>タグの範囲中に記述します。

■JavaScriptをHTMLに記述するタグ

```
<script> ～ </script>
```

またJavaScriptでは、alert("メッセージ")の命令で「メッセージ」を表示するダイアログボックスが表示されます。これらを利用して、P.438の「kantan.html」を次のように動作させるには、どのような内容を送信すればよいか考えてください。

JavaScriptが実行されるクライアントの環境で、一般ユーザーが「kantan.html」にアクセスすると、メッセージのダイアログボックスが表示される

解答例

問題1

次のように、メッセージを入力用のテキストボックスに入力し、[メッセージ送信]ボタンをクリックします。

```
<script>alert("メッセージ")</script>
```

FIG 20-08 次のように入力

FIG 20-09 一般ユーザーが、スクリプトが実行される環境でアクセスすると…

ブラウザが「アクティブスクリプトを無効」にする設定にしてある時など環境によっては動作しないこともあります。あらかじめ、ご了解ください。