ソースコード脆弱性診断 報告書

見本　Damn Vulnerable Web Application

1. 調査の概要

対象	Damn Vulnerable Web Application、略称 DVWA。PHP と MySQL で作られたウェブアプリケーションです
入手元	GitHub の digininja/DVWA、master ブランチのアーカイブ
対象の版	コミット `a1978a2fed195acd093010af6395f86a281537ca`
調査日	2026年10月23日
調査範囲	34ファイル、2,772行。行数は空行とコメントを含みます
調査方法	AI がソースコードを読んで脆弱性の候補を挙げ、担当者が該当するファイルと行を開いて1件ずつ確認しました。本番のサイトにはアクセスしていません。DVWA を起動して試すこともしていません
報告者	Scioloop

この報告書は、Scioloop がお客さまにお渡しする報告書の見本です。対象の DVWA は、セキュリティの学習のためにわざと脆弱性を入れて公開されているアプリケーションです。

調査範囲の内訳

区分	ファイル数	行数	主な中身
共通部分	16	1,628	ログイン、セッション、DB への接続、初期設定、設定ファイル、Docker の設定
練習ページの low 版	15	752	ログインの総当たり、SQL インジェクション、反射型と保存型の XSS、ファイルのアップロード、権限の回避
API の認証部分	3	392	トークンの作成と確認、API のログイン
合計	34	2,772	

ファイルごとの行数は付録に載せています。10,000行までの診断に収まる量です。

DVWA は同じ機能を low・medium・high・impossible の4段階で持っています。今回は low の版を調べ、impossible の版は修正方法の手本として参照しました。リポジトリ全体の PHP と JavaScript は178ファイル、13,641行あります。見本のため、Scioloop の調査項目に当たる機能に絞りました。次のものは範囲に入れていません。

- medium・high・impossible の版。impossible の版のうち6ファイル、294行は修正方法の手本として読みました
- ほかの練習ページ。コマンド実行、CSRF、ファイルの読み込み、CAPTCHA、弱いセッション ID、DOM 型の XSS、CSP、JavaScript、オープンリダイレクト、暗号、アクセス制御、API のほかの部分
- 外部のライブラリの `dvwa/includes/Parsedown.php` と `external/recaptcha`
- テスト、文書、画像、MySQL 以外の DB 向けの SQL

2. まとめ

深刻度の決め方

深刻度	基準
高	ログインしていない人や一般の利用者が、特別な条件なしに、ほかの利用者の情報を読む・書き換える・消す、またはサーバーでプログラムを動かすことにつながるもの
中	起きたときの被害は大きいものの、ソースコードを見られる、何度も試せる、別の脆弱性と組み合わさる、といった条件が重なって起きるもの
低	単独では被害が小さく、ほかの対策を厚くするためのもの

深刻度ごとの件数

深刻度	件数
高	4
中	3
低	0
合計	7

指摘の一覧

番号	種類	題	深刻度
1	ファイルのアップロード	アップロードされたファイルの種類を確認していない	高
2	SQL インジェクション	入力をそのまま SQL 文に組み込んでいる	高
3	権限の確認漏れ	管理者だけの操作で権限を確認していない	高
4	クロスサイトスクリプティング	入力を変換せずに画面に表示している	高
5	認証情報のハードコード	鍵とパスワードがコードに書かれている	中
6	総当たり攻撃への対策	ログインの失敗に回数の制限がない	中
7	パスワードの保存方式	パスワードを MD5 で保存している	中

最初に直すべき3件

1. 指摘1のファイルのアップロード。サーバーでプログラムを動かされると、DB を含むサーバー上のすべてに影響します。直す場所は1ファイルで、impossible の版に手本もあります。
2. 指摘2の SQL インジェクション。利用者の情報やパスワードのハッシュを含む DB の中身を読まれるおそれがあります。指摘7の MD5 の問題も、DB を読まれたときに被害として表に出ます。

3. 指摘3の権限の確認漏れ。ログインしていない人でも DB を作り直せます。作り直すと、それまでのデータが消え、管理者のパスワードも初期の値に戻ります。

3. 指摘事項

指摘1　アップロードされたファイルの種類を確認していない

種類	ファイルのアップロード
深刻度	高
理由	ログインした利用者なら誰でも、特別な条件なしにサーバーへ任意のファイルを置けます。置いたファイルはサーバー上で動くおそれがあります

該当箇所

- `vulnerabilities/upload/source/low.php` 5〜6行。送られてきたファイル名をそのまま使い、公開フォルダの `hackable/uploads/` に保存先を決めています
- 同じファイルの9行。ファイルの種類、大きさ、中身を確認せずに `move_uploaded_file` で保存しています
- 同じファイルの15行。保存先のパスを画面に表示しています
- `vulnerabilities/upload/index.php` 52行。フォームの `MAX_FILE_SIZE` は送る側が書き換えられる値なので、サーバー側の上限の代わりにはなりません

起きうること

画像のほかに、サーバーで実行されるスクリプトのファイルもそのまま公開フォルダに置かれます。そのファイルにブラウザからアクセスされると、ウェブサーバーの設定によってはサーバー上でスクリプトが動きます。その場合、設定ファイルや DB の中身を読まれたり、サーバーを乗っ取られたりするおそれがあります。サーバー側に大きさの上限がないため、大きなファイルを続けて送られるとディスクが埋まります。

修正方法

impossible の版の `vulnerabilities/upload/source/impossible.php` 18〜43行が手本になります。要点は次の4つです。

- 中身を読んで、JPEG と PNG だけを受け付けます
- サーバー側で大きさの上限を確認します
- 保存する名前は送られてきた名前を使わず、ランダムに付け直します
- 画像を作り直して保存し、画像の中に紛れたデータを落とします。impossible の版は31〜40行でこれをしています

あわせて、アップロード先のフォルダではスクリプトを実行しない設定にするか、公開フォルダの外に保存することをおすすめします。

差分の例

```
--- a/vulnerabilities/upload/source/low.php
+++ b/vulnerabilities/upload/source/low.php
@@ -1,18 +1,34 @@
 <?php
```

```
 if( isset( $_POST[ 'Upload' ] ) ) {
-   // Where are we going to be writing to?
-   $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
-   $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
+   $tmp     = $_FILES[ 'uploaded' ][ 'tmp_name' ];
+   $size    = $_FILES[ 'uploaded' ][ 'size' ];
+   $info    = @getimagesize( $tmp );
+   $allowed = array( IMAGETYPE_JPEG => 'jpg', IMAGETYPE_PNG => 'png' );

-   // Can we move the file to the upload folder?
-   if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
-       // No
-       $html .= '<pre>Your image was not uploaded.</pre>';
+   // 種類は中身から判定する。送られてきた名前と Content-Type は信用しない
+   if( $size <= 0 || $size >= 100000 || $info === false || !isset( $allowed[ $info[2] ] ) ) {
+       $html .= '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';
    }
    else {
-       // Yes!
-       $html .= "<pre>{$target_path} succesfully uploaded!</pre>";
+       // 名前はランダムに付け直し、画像を作り直して中に紛れたデータを落とす
+       $name = bin2hex( random_bytes( 16 ) ) . '.' . $allowed[ $info[2] ];
+       $dest = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/' . $name;
+       if( $info[2] == IMAGETYPE_JPEG ) {
+           $img = imagecreatefromjpeg( $tmp );
+           $ok  = $img && imagejpeg( $img, $dest, 90 );
+       }
+       else {
+           $img = imagecreatefrompng( $tmp );
+           $ok  = $img && imagepng( $img, $dest, 9 );
+       }
+
+       if( $ok ) {
+           $html .= "<pre>{$name} succesfully uploaded!</pre>";
+       }
+       else {
+           $html .= '<pre>Your image was not uploaded.</pre>';
+       }
    }
 }
```

確認方法

- テスト環境で、通常の JPEG と PNG が保存され、ファイル名がランダムな名前になることを確認します
- 拡張子が .jpg でも中身が画像でないファイルは、断られることを確認します
- 100,000バイト以上のファイルが断られることを確認します
- アップロード先のフォルダでスクリプトが実行されない設定になっていることを、ウェブサーバーの設定ファイルで確認します

指摘2　入力をそのまま SQL 文に組み込んでいる

種類	SQL インジェクション
深刻度	高
理由	ログインした利用者なら誰でも、特別な条件なしに DB の中身を読めるおそれがあります

該当箇所

- `vulnerabilities/sqli/source/low.php` 5行。利用者 ID を `$_REQUEST` から受け取り、確認せずに使っています
- 同じファイルの10行。その値を引用符で囲み、SQL 文の文字列に直接つないでいます。MySQL 向けの処理です
- 同じファイルの31行と34行。SQLite 向けの処理も同じ形です
- 同じファイルの11行と36行。DB のエラーの中身をそのまま画面に表示しています
- 同じ形の箇所が `vulnerabilities/brute/source/low.php` 12行と `vulnerabilities/authbypass/change_user_details.php` 47行にもあります

起きうること

ID の欄に SQL の文法として解釈される文字を含めると、検索の条件が書き換わります。ほかの利用者の名前に加えて、パスワードのハッシュなど画面に出す予定のない列や、別の表の内容まで画面に出るおそれがあります。DB のエラーがそのまま表示されるので、表の作りを知る手がかりにもなります。brute の12行ではログインの判定そのものを、change_user_details.php の47行では書き換える対象を変えられるおそれがあります。

修正方法

SQL 文と値を分けて渡すプレースホルダーを使います。impossible の版の `vulnerabilities/sqli/source/impossible.php` 11〜18行が手本です。数字であることを確認してから整数にし、PDO のプレースホルダーで渡しています。

下の差分は、mysqli のままプレースホルダーに置き換える例です。後ろの処理を変えずに済みます。エラーの中身は画面に出さず、サーバーのログに残します。brute と change_user_details.php の該当行も同じ方法で直します。

差分の例

```
--- a/vulnerabilities/sqli/source/low.php
+++ b/vulnerabilities/sqli/source/low.php
@@ -3,12 +3,16 @@
 if( isset( $_REQUEST[ 'Submit' ] ) ) {
     // Get input
     $id = $_REQUEST[ 'id' ];
+    // 利用者の ID は数字だけ。それ以外はどの行にも当たらない 0 にする
+    $id = ctype_digit( (string) $id ) ? intval( $id ) : 0;

     switch ($_DVWA['SQLI_DB']) {
         case MYSQL:
             // Check database
-            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
-            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );
+            $stmt = mysqli_prepare( $GLOBALS["___mysqli_ston"], 'SELECT first_name, last_name FROM users WHERE
user_id = ?' );
```

```
+            mysqli_stmt_bind_param( $stmt, 'i', $id );
+            mysqli_stmt_execute( $stmt );
+            $result = mysqli_stmt_get_result( $stmt );

             // Get results
             while( $row = mysqli_fetch_assoc( $result ) ) {
@@ -28,12 +32,13 @@
             #$sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
             #$sqlite_db_connection->enableExceptions(true);

-            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
-            #print $query;
             try {
-                $results = $sqlite_db_connection->query($query);
+                $stmt = $sqlite_db_connection->prepare( 'SELECT first_name, last_name FROM users WHERE user_id =
:id;' );
+                $stmt->bindValue( ':id', $id, SQLITE3_INTEGER );
+                $results = $stmt->execute();
             } catch (Exception $e) {
-                echo 'Caught exception: ' . $e->getMessage();
+                // エラーの中身は画面に出さず、サーバーのログにだけ残す
+                error_log( $e->getMessage() );
                 exit();
             }
```

確認方法

- テスト環境で、ID に数字を入れたときの結果が修正の前と同じことを確認します
- 数字以外の文字を入れたとき、DB のエラーが表示されず、結果が0件になることを確認します
- 次のコマンドで、SQL 文に変数をつないでいる行を一覧にします。出てきた行は1件ずつ、プレースホルダーになっているかを確認します

```
grep -rnE '(SELECT|INSERT|UPDATE|DELETE).*\$' --include='*.php' .
```

指摘3　管理者だけの操作で権限を確認していない

種類	権限の確認漏れ
深刻度	高
理由	一般の利用者や、ログインしていない人でも、管理者だけに許すべき操作ができるためです

該当箇所

- `vulnerabilities/authbypass/get_user_data.php` 10〜13行。管理者かどうかの確認は、難易度が high と impossible のときだけです。このファイルではログインの確認もしていません
- `vulnerabilities/authbypass/change_user_details.php` 11〜14行。管理者かどうかの確認は impossible のときだけで、ログインの確認もありません。CSRF トークンも確認していません
- `vulnerabilities/authbypass/source/low.php` は中身がコメントだけで、管理者かどうかの確認がありません。`vulnerabilities/authbypass/index.php` 6行はログインだけを確認しています
- `dvwa/includes/dvwaPage.inc.php` 311〜313行。管理者でない人にはメニューを表示していません。表示を隠しているだけで、アクセスの制限にはなっていません
- `setup.php` 6行。ログインを求めていません。12〜34行で DB を作り直す処理を受け付けます。20行で CSRF トークンを確認していますが、トークンはページを開いた人なら誰でも受け取れます
- `dvwa/includes/DBMS/MySQL.php` 22行と55〜59行。作り直すときは DB を消してから、初期のアカウントを初期のパスワードで作ります

起きうること

管理者でない利用者が、全利用者の ID と名前の一覧を取り出し、名前を書き換えられます。low のときは、ログインしていない人でも同じことができます。セットアップのページはログインなしで使えるため、誰でも DB を消して作り直せます。作り直すと、それまでのデータは消え、管理者のパスワードは初期の値に戻ります。

修正方法

- 管理者だけに許す処理では、処理の最初でログインと管理者であることをサーバー側で確認します。難易度などの条件で確認を外さないようにします。手本は `vulnerabilities/authbypass/source/impossible.php` 8〜12行です
- メニューを隠すのは見た目のためと考え、アクセスの制限は各処理の中で行います
- change_user_details.php には、権限の確認とあわせて CSRF トークンの確認を入れます
- セットアップのページは、初回のセットアップが済んだあとは管理者だけが使えるようにします。本番の環境には置かないことをおすすめします
- users の表には role の列があり、`dvwa/includes/DBMS/MySQL.php` 67行で管理者に admin を入れています。名前が admin かどうかで判定するより、この列で判定するほうが安全です

下の差分は、2つの処理に確認を入れ、セットアップのページを初回のあとは管理者だけに限る例です。

差分の例

```
--- a/vulnerabilities/authbypass/get_user_data.php
+++ b/vulnerabilities/authbypass/get_user_data.php
@@ -2,12 +2,12 @@
```

```
 define( 'DVWA_WEB_PAGE_TO_ROOT', '../../' );
 require_once DVWA_WEB_PAGE_TO_ROOT . 'dvwa/includes/dvwaPage.inc.php';

+dvwaPageStartup( array( 'authenticated' ) );
 dvwaDatabaseConnect();

-/*
-On high and impossible, only the admin is allowed to retrieve the data.
-*/
-if ((dvwaSecurityLevelGet() == "high" || dvwaSecurityLevelGet() == "impossible") && dvwaCurrentUser() !=
"admin") {
+// 難易度に関係なく、管理者だけに返す
+if (dvwaCurrentUser() != "admin") {
+   http_response_code(403);
     print json_encode (array ("result" => "fail", "error" => "Access denied"));
     exit;
 }
--- a/vulnerabilities/authbypass/change_user_details.php
+++ b/vulnerabilities/authbypass/change_user_details.php
@@ -2,13 +2,12 @@
 define( 'DVWA_WEB_PAGE_TO_ROOT', '../../' );
 require_once DVWA_WEB_PAGE_TO_ROOT . 'dvwa/includes/dvwaPage.inc.php';

+dvwaPageStartup( array( 'authenticated' ) );
 dvwaDatabaseConnect();

-/*
-On impossible only the admin is allowed to retrieve the data.
-*/
-
-if (dvwaSecurityLevelGet() == "impossible" && dvwaCurrentUser() != "admin") {
+// 難易度に関係なく、管理者だけに変更を許す
+if (dvwaCurrentUser() != "admin") {
+   http_response_code(403);
     print json_encode (array ("result" => "fail", "error" => "Access denied"));
     exit;
 }
--- a/setup.php
+++ b/setup.php
@@ -10,6 +10,12 @@
 $page[ 'page_id' ] = 'setup';

 if( isset( $_POST[ 'create_db' ] ) ) {
+   // 初回のセットアップが済んだあとは、管理者でログインしているときだけ作り直せる
+   if( dvwaIsInstalled() && dvwaCurrentUser() != 'admin' ) {
+       dvwaMessagePush( 'Only the admin user can reset the database.' );
+       dvwaPageReload();
+   }
+
     // Anti-CSRF
     if (array_key_exists ("session_token", $_SESSION)) {
         $session_token = $_SESSION[ 'session_token' ];
--- a/dvwa/includes/dvwaPage.inc.php
+++ b/dvwa/includes/dvwaPage.inc.php
@@ -596,6 +596,24 @@
     }
 }

+// users の表があれば、初回のセットアップは済んでいる。login.php 36～44行と同じ判定
+function dvwaIsInstalled() {
+   global $_DVWA;
+
+   try {
```

```
+       $link = mysqli_connect( $_DVWA[ 'db_server' ], $_DVWA[ 'db_user' ], $_DVWA[ 'db_password' ], '', $_DVWA[
'db_port' ] );
+       $stmt = mysqli_prepare( $link, "SELECT 1 FROM information_schema.tables WHERE table_schema = ? AND
table_name = 'users' LIMIT 1" );
+       mysqli_stmt_bind_param( $stmt, 's', $_DVWA[ 'db_database' ] );
+       mysqli_stmt_execute( $stmt );
+       $installed = ( mysqli_stmt_fetch( $stmt ) === true );
+       mysqli_close( $link );
+       return $installed;
+   } catch( Throwable $e ) {
+       // 判定できないときは、済んでいるものとして扱う
+       return true;
+   }
+}
+
 // -- END (Database Management)
```

確認方法

- テスト環境で、管理者でない利用者でログインし、get_user_data.php と change_user_details.php が 403 を返すことを確認します
- ログインしていない状態では、ログインのページへ移ることを確認します
- 管理者でログインしたときは、一覧の表示と名前の変更がこれまでどおりできることを確認します
- セットアップが済んだ環境で、ログインしていない状態と管理者でない利用者の状態では、DB を作り直せないことを確認します

指摘4　入力を変換せずに画面に表示している

種類	クロスサイトスクリプティング、略称 XSS
深刻度	高
理由	ゲストブックに書いた内容が、ページを開いたほかの利用者のブラウザでスクリプトとして動くおそれがあるためです。low ではセッションの Cookie に HttpOnly が付いていないので、ログインの状態を盗まれる危険が大きくなります

該当箇所

- `vulnerabilities/xss_s/source/low.php` 5〜6行と9〜16行。名前とメッセージを、SQL 用の変換だけをして保存しています。HTML 用の変換はしていません
- `dvwa/includes/dvwaPage.inc.php` 616〜625行。ゲストブックを表示するとき、HTML 用の変換を impossible のときだけ行い、それ以外ではそのまま出しています
- `vulnerabilities/xss_r/source/low.php` 8行。URL の name の値をそのまま画面に出しています
- `vulnerabilities/authbypass/authbypass.js` 43〜49行。利用者の名前を innerHTML で組み立てています。名前は指摘3の問題で誰でも書き換えられます
- `dvwa/includes/dvwaPage.inc.php` 54〜57行。low では、セッションの Cookie に HttpOnly を付けていません

起きうること

ゲストブックの名前やメッセージに、HTML やスクリプトとして解釈される文字列を含めると、そのページを開いたほかの利用者のブラウザで実行されます。ログインの状態を盗まれたり、その利用者になりすまして操作されたりするおそれがあります。xss_r の8行では、細工した URL を開かせることで同じことが起きます。authbypass.js では、名前を書き換えられると、管理者が利用者の一覧を開いたときに実行されます。

修正方法

画面に出す直前に、`htmlspecialchars` で HTML 用に変換します。`ENT_QUOTES` と文字コードを指定します。手本は impossible の版の `vulnerabilities/xss_r/source/impossible.php` 9行と、`dvwa/includes/dvwaPage.inc.php` 617〜618行です。

保存型の impossible の版 `vulnerabilities/xss_s/source/impossible.php` は、14行と19行で保存する前に変換しています。この方式は、同じデータを API や別の画面で使うときに変換が漏れたり二重になったりしやすいため、表示する側での変換をおすすめします。JavaScript では innerHTML で組み立てるのをやめ、要素を作って値を入れます。あわせて、どの難易度でもセッションの Cookie に HttpOnly を付けます。

差分の例

```
--- a/dvwa/includes/dvwaPage.inc.php
+++ b/dvwa/includes/dvwaPage.inc.php
@@ -52,8 +52,9 @@
        $samesite = "Strict";
    }
    else {
-       $httponly = false;
-       $samesite = "";
+       // スクリプトからセッションの Cookie を読めないようにする
```

```
+       $httponly = true;
+       $samesite = "Lax";
    }

    $maxlifetime = 86400;
@@ -613,14 +614,9 @@
    $guestbook = '';

    while( $row = mysqli_fetch_row( $result ) ) {
-       if( dvwaSecurityLevelGet() == 'impossible' ) {
-           $name    = htmlspecialchars( $row[0] );
-           $comment = htmlspecialchars( $row[1] );
-       }
-       else {
-           $name    = $row[0];
-           $comment = $row[1];
-       }
+       // 表示するときに必ず変換する。保存するときの変換には頼らない
+       $name    = htmlspecialchars( $row[0], ENT_QUOTES, 'UTF-8' );
+       $comment = htmlspecialchars( $row[1], ENT_QUOTES, 'UTF-8' );

        $guestbook .= "<div id=\"guestbook_comments\">Name: {$name}<br />" . "Message: {$comment}<br /></div>\n";
    }
--- a/vulnerabilities/xss_r/source/low.php
+++ b/vulnerabilities/xss_r/source/low.php
@@ -5,7 +5,7 @@
 // Is there any input?
 if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
-   $html .= '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
+   $html .= '<pre>Hello ' . htmlspecialchars( $_GET[ 'name' ], ENT_QUOTES, 'UTF-8' ) . '</pre>';
 }

 ?>
--- a/vulnerabilities/authbypass/authbypass.js
+++ b/vulnerabilities/authbypass/authbypass.js
@@ -37,16 +37,31 @@
        table_body = document.getElementById('user_table').getElementsByTagName('tbody')[0];
        users.forEach(updateTable);

+       // 名前は innerHTML で組み立てず、入力欄の値として入れる
+       function textInput (id, name, value) {
+           var input = document.createElement('input');
+           input.type = 'text';
+           input.id = id;
+           input.name = name;
+           input.value = value;
+           return input;
+       }
+
        function updateTable (user) {
+           var id = user['user_id'];
            var row = table_body.insertRow(0);
            var cell0 = row.insertCell(-1);
-           cell0.innerHTML = user['user_id'] + '<input type="hidden" id="user_id_' + user['user_id'] + '"
name="user_id" value="' + user['user_id'] + '" />';
+           cell0.textContent = id;
            var cell1 = row.insertCell(1);
-           cell1.innerHTML = '<input type="text" id="first_name_' + user['user_id'] + '" name="first_name"
value="' + user['first_name'] + '" />';
+           cell1.appendChild(textInput('first_name_' + id, 'first_name', user['first_name']));
            var cell2 = row.insertCell(2);
```

```
-           cell2.innerHTML = '<input type="text" id="surname_' + user['user_id'] + '" name="surname" value="' +
user['surname'] + '" />';
+           cell2.appendChild(textInput('surname_' + id, 'surname', user['surname']));
            var cell3 = row.insertCell(3);
-           cell3.innerHTML = '<input type="button" value="Update" onclick="submit_change(' + user['user_id'] +
')" />';
+           var button = document.createElement('input');
+           button.type = 'button';
+           button.value = 'Update';
+           button.addEventListener('click', function () { submit_change(id); });
+           cell3.appendChild(button);
        }
    };
    xhr.send();
```

確認方法

- テスト環境で、ゲストブックと name の欄に HTML の記号を含む文字列を入れ、画面に記号のまま表示されることを確認します。ブラウザの開発者ツールでは、記号が `<` などに変換されていることを確認します
- 利用者の名前に同じ文字列を入れ、管理者の一覧で記号のまま表示されることを確認します
- ログインしたあと、ブラウザの開発者ツールで、セッションの Cookie に HttpOnly が付いていることを確認します

指摘5　鍵とパスワードがコードに書かれている

種類	認証情報のハードコード
深刻度	中
理由	ソースコードや設定の写しを見られた時点で、API のトークンを自由に作れるようになり、DB のパスワードも知られます。ソースコードを見られることが前提なので中としました。ソースコードを公開している場合や外部に渡している場合は、高として扱ってください

該当箇所

鍵とパスワードの値は、この報告書では伏せています。

- `vulnerabilities/api/src/Token.php` 10行。API のトークンを暗号化する鍵を定数で書いています。鍵の長さも、9行で指定している AES-128 に必要な16バイトに足りません
- `vulnerabilities/api/src/Login.php` 10行と12行。トークンの中に入れる値を定数で書いています
- `vulnerabilities/api/src/LoginController.php` 60行、86行、96行。API の利用者名とパスワード、クライアントの秘密の値を、コードの中の文字列と比べています
- `config/config.inc.php.dist` 21行。環境変数 DB_PASSWORD がないときに、コードに書いた DB のパスワードで接続します。`compose.yml` 31行と34行にも DB のパスワードが書かれています
- `dvwa/includes/DBMS/MySQL.php` 125〜133行。セットアップのたびに、設定ファイルの写し `config/config.inc.php.bak` を作ります。拡張子が .php でないため、ウェブサーバーの設定によっては中の DB のパスワードがそのまま読めます
- `dvwa/includes/DBMS/MySQL.php` 55〜59行と `setup.php` 105行。初期のアカウントのパスワードがコードに書かれ、管理者のものはセットアップの画面にも表示されます

起きうること

ソースコードを読んだ人は、正しい API のトークンを自分で作れます。トークンの確認は `vulnerabilities/api/src/Login.php` 33行と43行で、復号できることと、中の値がコードの定数と一致することと、期限だけを見ているため、作ったトークンはそのまま通ります。DB のパスワードは、設定ファイルの写しや compose.yml から知られるおそれがあります。初期のアカウントのパスワードは公開されている値なので、変えずに使っていると誰でもログインできます。鍵をコードから消しても、git の履歴には残ります。

修正方法

- 鍵とパスワードは環境変数や秘密情報の管理サービスから読み、コードと設定の見本には書きません。値がないときは既定の値で動かさず、エラーで止めます
- API のトークンを暗号化する鍵は、16バイトの乱数から作ります
- 設定ファイルの写しを作る処理を消し、設定ファイルは公開フォルダの外に置きます
- 初期のアカウントは、最初のログインでパスワードの変更を求めるか、セットアップのときに乱数で作って一度だけ表示します
- コードに書かれていた鍵とパスワードは漏れたものとして扱い、すべて新しい値に変えます

差分の例

```
--- a/vulnerabilities/api/src/Token.php
```

```
+++ b/vulnerabilities/api/src/Token.php
@@ -7,7 +7,15 @@
 #[OAT\Schema(required: ['token'])]
 class Token {
     private const ENCRYPTION_CIPHER = "aes-128-gcm";
-    private const ENCRYPTION_KEY = "********";
+
+    // 鍵はコードに書かず、環境変数 API_TOKEN_KEY から読む。値は 16 バイトの乱数を base64 にしたもの
+    private static function key() {
+        $key = base64_decode( (string) getenv( 'API_TOKEN_KEY' ), true );
+        if( $key === false || strlen( $key ) !== 16 ) {
+            throw new \RuntimeException( 'API_TOKEN_KEY is not set correctly.' );
+        }
+        return $key;
+    }

     # Not sure if this is needed
     #[OAT\Property(example: "11111")]
@@ -22,7 +30,7 @@
     private static function encrypt($cleartext) {
         $ivlen = openssl_cipher_iv_length(self::ENCRYPTION_CIPHER);
         $iv = openssl_random_pseudo_bytes($ivlen);
-        $ciphertext = openssl_encrypt($cleartext, self::ENCRYPTION_CIPHER, self::ENCRYPTION_KEY, $options=0, $iv,
$tag);
+        $ciphertext = openssl_encrypt($cleartext, self::ENCRYPTION_CIPHER, self::key(), $options=0, $iv, $tag);
         $ret = base64_encode ($tag . ":::::" . $iv . ":::::" . $ciphertext);
         return $ret;
     }
@@ -36,7 +44,7 @@
         $value = $bits[2];
         $iv = $bits[1];
         $tag = $bits[0];
-        $cleartext = openssl_decrypt($value, self::ENCRYPTION_CIPHER, self::ENCRYPTION_KEY, $options=0, $iv,
$tag);
+        $cleartext = openssl_decrypt($value, self::ENCRYPTION_CIPHER, self::key(), $options=0, $iv, $tag);
         return $cleartext;
     }
     public function create_token($secret, $expires) {
--- a/config/config.inc.php.dist
+++ b/config/config.inc.php.dist
@@ -18,7 +18,10 @@
 $_DVWA[ 'db_server' ]   = getenv('DB_SERVER') ?: '127.0.0.1';
 $_DVWA[ 'db_database' ] = getenv('DB_DATABASE') ?: 'dvwa';
 $_DVWA[ 'db_user' ]     = getenv('DB_USER') ?: 'dvwa';
-$_DVWA[ 'db_password' ] = getenv('DB_PASSWORD') ?: '********';
+$_DVWA[ 'db_password' ] = getenv('DB_PASSWORD');
+if( $_DVWA[ 'db_password' ] === false || $_DVWA[ 'db_password' ] === '' ) {
+    die( 'DB_PASSWORD is not set.' );
+}
 $_DVWA[ 'db_port']      = getenv('DB_PORT') ?: '3306';

 # ReCAPTCHA settings
--- a/dvwa/includes/DBMS/MySQL.php
+++ b/dvwa/includes/DBMS/MySQL.php
@@ -122,16 +122,6 @@
 }
 dvwaMessagePush( "Data inserted into 'guestbook' table." );

-// Copy .bak for a fun directory listing vuln
-$conf = DVWA_WEB_PAGE_TO_ROOT . 'config/config.inc.php';
-$bakconf = DVWA_WEB_PAGE_TO_ROOT . 'config/config.inc.php.bak';
-if (file_exists($conf)) {
-    // Who cares if it fails. Suppress.
```

```
-    @copy($conf, $bakconf);
-}
-
-dvwaMessagePush( "Backup file /config/config.inc.php.bak automatically created" );
-
 // Done
 dvwaMessagePush( "<em>Setup successful</em>!" );
```

確認方法

- 環境変数 API_TOKEN_KEY と DB_PASSWORD をわざと空にして起動し、エラーで止まることを確認します。設定が効いていることも、この方法で確認できます
- gitleaks などの秘密情報を検出するツールで、作業中のファイルと git の履歴の両方を調べ、検出が0件になることを確認します
- セットアップのあと、config のフォルダに .bak のファイルができていないことを確認します
- 鍵を変えたあと、変える前に発行したトークンが受け付けられないことを確認します

指摘6　ログインの失敗に回数の制限がない

種類	総当たり攻撃への対策
深刻度	中
理由	パスワードを何度でも試せるため、弱いパスワードの利用者は自動の試行で破られるおそれがあります。成功までに多くの試行が必要なので中としました

該当箇所

- `login.php` 10〜57行。アプリケーション全体のログインです。失敗の回数を数えておらず、待ち時間もロックもありません。この処理は難易度に関係なく使われます
- `vulnerabilities/brute/source/low.php` 3〜27行。練習ページのログインも同じです
- 同じファイルの3行、5行、8行と、`vulnerabilities/brute/index.php` 15行と42行。low ではパスワードを GET で送るため、パスワードが URL に入り、ウェブサーバーのアクセスログやブラウザの履歴に残ります
- `dvwa/includes/DBMS/MySQL.php` 55〜59行。初期のアカウントに、よく使われる短いパスワードを設定しています

起きうること

よく使われるパスワードの一覧を使って、自動で何千回もログインを試されます。止める仕組みがないため、弱いパスワードの利用者から順に破られるおそれがあります。

修正方法

- 失敗が続いたアカウントは一定時間ロックします。手本は `vulnerabilities/brute/source/impossible.php` 18〜50行と87〜90行です。失敗の回数を DB に記録し、3回失敗すると15分ロックしています
- impossible の版は、ロック中とパスワードの誤りとで同じメッセージを返しています。85行です。どのアカウントがあるかを知られないために、この形を残します
- アカウントごとの制限に加えて、IP アドレスごとの制限もおすすめします。1つの IP アドレスから多くのアカウントを試す攻撃は、アカウントごとの制限では止まりません
- パスワードは POST で送ります
- impossible の版は、失敗のたびに82行で2〜4秒待たせています。この方法は待っている間サーバーの処理を1つふさぐため、利用者が多いときは待たせずに断る形をおすすめします

下の差分は、login.php にアカウントごとのロックとプレースホルダーを入れる例です。パスワードの比べ方は指摘7で直します。

差分の例

```
--- a/login.php
+++ b/login.php
@@ -17,14 +17,9 @@

    checkToken( $_REQUEST[ 'user_token' ], $session_token, 'login.php' );

+   // 値はプレースホルダーで渡すので、ここではエスケープしない
    $user = $_POST[ 'username' ];
-   $user = stripslashes( $user );
-   $user = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $user ) : ((trigger_error("[MySQLConverterToo] Fix the
mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
```

```
-
    $pass = $_POST[ 'password' ];
-   $pass = stripslashes( $pass );
-   $pass = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass ) : ((trigger_error("[MySQLConverterToo] Fix the
mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
-   $pass = md5( $pass );

    // Validate that username and password are not empty (improved login validation).
    if( trim( $user ) === '' || trim( $_POST[ 'password' ] ) === '' ) {
@@ -43,12 +38,26 @@
        dvwaRedirect( DVWA_WEB_PAGE_TO_ROOT . 'setup.php' );
    }

-   $query  = "SELECT * FROM `users` WHERE user='$user' AND password='$pass';";
-   $result = @mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '.<br />Try <a href="setup.php">installing again</a>.</pre>'
);
-   if( $result && mysqli_num_rows( $result ) == 1 ) {    // Login Successful...
-       dvwaMessagePush( "You have logged in as '{$user}'" );
-       dvwaLogin( $user );
+   // 失敗が続いたアカウントは一定時間ロックする。手本は vulnerabilities/brute/source/impossible.php 18〜50行
+   $max_failed = 5;
+   $lock_min   = 15;
+
+   $stmt = $db->prepare( 'SELECT user, password, failed_login, last_login FROM users WHERE user = :user LIMIT 1'
);
+   $stmt->execute( array( ':user' => $user ) );
+   $row = $stmt->fetch();
+
+   $locked = $row && $row[ 'failed_login' ] >= $max_failed
+       && time() < strtotime( $row[ 'last_login' ] ) + $lock_min * 60;
+
+   if( $row && !$locked && hash_equals( $row[ 'password' ], md5( $pass ) ) ) {    // Login Successful...
+       $db->prepare( 'UPDATE users SET failed_login = 0, last_login = NOW() WHERE user = :user' )->execute(
array( ':user' => $row[ 'user' ] ) );
+       dvwaMessagePush( "You have logged in as '" . htmlspecialchars( $row[ 'user' ] ) . "'" );
+       dvwaLogin( $row[ 'user' ] );
        dvwaRedirect( DVWA_WEB_PAGE_TO_ROOT . 'index.php' );
+   }
+
+   if( $row ) {
+       $db->prepare( 'UPDATE users SET failed_login = failed_login + 1, last_login = NOW() WHERE user = :user'
)->execute( array( ':user' => $row[ 'user' ] ) );
    }

    // Login failed
```

確認方法

- テスト環境で、同じアカウントで5回続けて失敗したあと、正しいパスワードでもログインできないことを確認します
- 15分たつと、正しいパスワードでログインできることを確認します
- ログインに成功すると、失敗の回数が0に戻ることを確認します
- ロック中のメッセージが、パスワードを誤ったときと同じであることを確認します

指摘7　パスワードを MD5 で保存している

種類	パスワードの保存方式
深刻度	中
理由	DB の中身が漏れたとき、多くのパスワードが短時間で元に戻されます。DB が漏れることが前提なので中としましたが、指摘2の SQL インジェクションがあるため、その前提はそろいやすい状態です

該当箇所

- `login.php` 27行。パスワードを MD5 にしてから DB の値と比べています
- `vulnerabilities/brute/source/low.php` 9行。練習ページも同じです
- `dvwa/includes/DBMS/MySQL.php` 42行。password の列は32文字で、MD5 の長さに合わせてあります
- `dvwa/includes/DBMS/MySQL.php` 55〜59行。初期のアカウントを MD5 で作っています
- `vulnerabilities/brute/source/impossible.php` 16行。手本の impossible の版も MD5 のままです。手本をそのまま使っても、この問題は直りません

起きうること

MD5 はとても速く計算できるハッシュで、ソルトもないため、同じパスワードの利用者は同じ値になります。よく使われるパスワードの MD5 の値は一覧として出回っており、DB が漏れると多くのパスワードがすぐに分かります。ほかのサービスで同じパスワードを使っている利用者は、そちらにも被害が広がります。

修正方法

- パスワードは `password_hash` で保存し、`password_verify` で確認します。ソルトが自動で付き、計算に時間のかかる方式になります
- password の列を255文字に広げます。今ある DB は ALTER TABLE で列を広げます
- 今の MD5 の値は、利用者が次に正しくログインしたときに新しい方式に置き換えます。長くログインしていない利用者には、パスワードの再設定を求めます
- `login.php` 25〜26行では、MD5 にする前にパスワードに stripslashes とエスケープをかけています。新しい方式では、入力をそのまま password_hash と password_verify に渡します

下の差分は、指摘6の修正を当てたあとの login.php に対する差分と、セットアップの差分です。初期のパスワードの値は伏せています。

差分の例

```
--- a/login.php
+++ b/login.php
@@ -49,7 +49,19 @@
    $locked = $row && $row[ 'failed_login' ] >= $max_failed
        && time() < strtotime( $row[ 'last_login' ] ) + $lock_min * 60;

-   if( $row && !$locked && hash_equals( $row[ 'password' ], md5( $pass ) ) ) {    // Login Successful...
+   $ok = false;
+   if( $row && !$locked ) {
+       if( password_verify( $pass, $row[ 'password' ] ) ) {
+           $ok = true;
+       }
```

```
+       elseif( strlen( $row[ 'password' ] ) == 32 && hash_equals( $row[ 'password' ], md5( $pass ) ) ) {
+           // 移行の間だけ古い MD5 を受け付け、ログインできた時点で新しい方式に置き換える
+           $ok = true;
+           $db->prepare( 'UPDATE users SET password = :hash WHERE user = :user' )->execute( array( ':hash' =>
password_hash( $pass, PASSWORD_DEFAULT ), ':user' => $row[ 'user' ] ) );
+       }
+   }
+
+   if( $ok ) {    // Login Successful...
        $db->prepare( 'UPDATE users SET failed_login = 0, last_login = NOW() WHERE user = :user' )->execute(
array( ':user' => $row[ 'user' ] ) );
        dvwaMessagePush( "You have logged in as '" . htmlspecialchars( $row[ 'user' ] ) . "'" );
        dvwaLogin( $row[ 'user' ] );
--- a/dvwa/includes/DBMS/MySQL.php
+++ b/dvwa/includes/DBMS/MySQL.php
@@ -39,7 +39,7 @@
    dvwaPageReload();
 }

-$create_tb = "CREATE TABLE users (user_id int(6),first_name varchar(15),last_name varchar(15), user varchar(15),
password varchar(32),avatar varchar(70), last_login TIMESTAMP, failed_login INT(3), role VARCHAR(20) DEFAULT
'user', account_enabled TINYINT(1) DEFAULT 1, PRIMARY KEY (user_id));";
+$create_tb = "CREATE TABLE users (user_id int(6),first_name varchar(15),last_name varchar(15), user varchar(15),
password varchar(255),avatar varchar(70), last_login TIMESTAMP, failed_login INT(3), role VARCHAR(20) DEFAULT
'user', account_enabled TINYINT(1) DEFAULT 1, PRIMARY KEY (user_id));";
 if( !mysqli_query($GLOBALS["___mysqli_ston"],  $create_tb ) ) {
    dvwaMessagePush( "Table could not be created<br />SQL: " . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false))
);
    dvwaPageReload();
@@ -52,11 +52,11 @@
 $avatarUrl  = $base_dir . 'hackable/users/';

 $insert = "INSERT INTO users (user_id, first_name, last_name, user, password, avatar, last_login, failed_login)
VALUES
-   ('1','admin','admin','admin',MD5('********'),'{$avatarUrl}admin.jpg', NOW(), '0'),
-   ('2','Gordon','Brown','gordonb',MD5('********'),'{$avatarUrl}gordonb.jpg', NOW(), '0'),
-   ('3','Hack','Me','1337',MD5('********'),'{$avatarUrl}1337.jpg', NOW(), '0'),
-   ('4','Pablo','Picasso','pablo',MD5('********'),'{$avatarUrl}pablo.jpg', NOW(), '0'),
-   ('5','Bob','Smith','smithy',MD5('********'),'{$avatarUrl}smithy.jpg', NOW(), '0');";
+   ('1','admin','admin','admin','" . password_hash( '********', PASSWORD_DEFAULT ) . "','{$avatarUrl}admin.jpg',
NOW(), '0'),
+   ('2','Gordon','Brown','gordonb','" . password_hash( '********', PASSWORD_DEFAULT ) .
"','{$avatarUrl}gordonb.jpg', NOW(), '0'),
+   ('3','Hack','Me','1337','" . password_hash( '********', PASSWORD_DEFAULT ) . "','{$avatarUrl}1337.jpg',
NOW(), '0'),
+   ('4','Pablo','Picasso','pablo','" . password_hash( '********', PASSWORD_DEFAULT ) .
"','{$avatarUrl}pablo.jpg', NOW(), '0'),
+   ('5','Bob','Smith','smithy','" . password_hash( '********', PASSWORD_DEFAULT ) . "','{$avatarUrl}smithy.jpg',
NOW(), '0');";
 if( !mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) ) {
    dvwaMessagePush( "Data could not be inserted into 'users' table<br />SQL: " .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) );
    dvwaPageReload();
```

確認方法

- 修正のあとにセットアップした DB で、password の列の値が `$2y$` で始まることを確認します
- 古い MD5 の値を持つ利用者が正しいパスワードで一度ログインすると、値が新しい方式に置き換わることを確認します
- ログインの処理で、移行のための1か所のほかに md5 を使っていないことを確認します

件数に入れなかった気づき

見本のため件数には入れていませんが、範囲の中で次の点も見つかりました。

- `dvwa/includes/dvwaPage.inc.php` 97〜105行。low では、ログインの前後でセッション ID を変えていません。セッション固定の攻撃を受けるおそれがあります
- `dvwa/includes/dvwaPage.inc.php` 60行。セッションの Cookie に、どの難易度でも Secure を付けていません。HTTPS で公開するときは付けます
- `dvwa/includes/dvwaPage.inc.php` 650行。CSRF トークンを `md5( uniqid() )` で作っています。uniqid は時刻から作る値なので推測されやすく、random_bytes で作るほうが安全です
- `dvwa/includes/dvwaPage.inc.php` 193〜194行と405行。Cookie の theme の値を、変換せずに HTML に出しています
- `vulnerabilities/xss_s/index.php` 16〜19行。ゲストブックを全部消す処理に、CSRF トークンの確認も権限の確認もありません
- `php.ini` 5行。allow_url_include を有効にしています。本番では無効にします
- `login.php` 47行、`vulnerabilities/brute/source/low.php` 13行、`vulnerabilities/xss_s/source/low.php` 17行。DB のエラーの中身を画面に表示しています

4. 問題が見つからなかった点

- `login.php` 22行、26行、46行。アプリケーション全体のログインは、値を SQL 用にエスケープして引用符で囲んでいます。この形では SQL インジェクションにつながる箇所は見つかりませんでした。指摘6と指摘7の修正では、より安全なプレースホルダーに置き換えます
- `login.php` 18行、`security.php` 15行、`setup.php` 20行。状態を変えるフォームでは CSRF トークンを確認しています。トークンの作り方の弱さは「件数に入れなかった気づき」に書きました
- `security.php` 18〜31行。難易度の値を決まった4つに限り、ほかの値は impossible として扱っています
- `instructions.php` 20〜24行。表示する文書を決まった一覧から選んでいます。ほかのファイルを読ませることはできません
- `vulnerabilities/upload/source/low.php` 6行。ファイル名から basename でフォルダの部分を落としているので、アップロード先のほかのフォルダには書き込めません。種類を確認していない点は指摘1に書きました
- `dvwa/includes/dvwaPage.inc.php` 577〜579行。PDO の接続で、エラーを例外にし、プレースホルダーを DB 側で処理する設定にしています。修正の土台として使えます
- `dvwa/includes/dvwaPage.inc.php` 685行。セットアップの画面では、DB のパスワードを伏せて表示しています
- `vulnerabilities/api/src/Token.php` 9行と23〜25行。トークンの暗号方式は AES-GCM で、IV を毎回乱数で作っています。方式と IV の作り方は適切で、問題は鍵の置き場所と長さだけです
- `compose.yml` 25行と28〜39行。アプリケーションのポートは 127.0.0.1 だけで受け付け、DB のポートは外に出していません
- `config/config.inc.php.dist` 33行。難易度の既定値は impossible です
- `phpinfo.php` 6行。サーバーの設定を表示するページは、ログインが必要です
- `vulnerabilities/xss_r/source/low.php` 3行の `X-XSS-Protection: 0` は、古いブラウザの XSS フィルターを止める指定です。今は 0 を指定するか付けないことが推奨されているため、この行は問題ありません

5. 今後の進め方

直す順番

順番	指摘	内容	時期の目安
1	指摘1	アップロードで種類と大きさを確認し、名前を付け直す	すぐに
2	指摘2	SQL 文をプレースホルダーに置き換え、DB のエラーを画面に出さない	すぐに
3	指摘3	管理者だけの処理に権限の確認を入れ、セットアップのページを閉じる	すぐに
4	指摘4	表示の直前に HTML 用に変換し、Cookie に HttpOnly を付ける	次のリリースまでに
5	指摘5	鍵とパスワードを環境変数に移し、すべて新しい値に変える	次のリリースまでに
6	指摘6・7	ログインのロックとパスワードの保存方式を、login.php でまとめて直す	計画して

指摘6と指摘7は login.php の同じ部分を直すため、まとめて進めると手戻りがありません。指摘5で鍵を変えるのは、コードから鍵を消したあとにします。先に変えると、新しい鍵もコードや履歴に残るおそれがあります。

再確認

修正が済んだら、修正した版のソースコードをお送りください。この報告書の7件について、該当箇所が直っているかと、同じ形の書き方がほかに残っていないかを確認し、結果を表にしてお返しします。

6. 注意事項

- DVWA は、セキュリティの学習のためにわざと脆弱性を入れて作られたアプリケーションです。README には、インターネットにつながるサーバーに置かないよう警告があります。この報告書の指摘の多くは、作者が意図して入れたものです
- この報告書は、Scioloop の報告書の形を示すための見本です。DVWA の作者に向けた報告ではありません。DVWA の SECURITY.md でも、意図した脆弱性なので報告しないよう求めています
- DVWA のライセンスは GNU General Public License のバージョン3以降です。この報告書に引用したコードと差分の例は DVWA のコードを元にしているため、同じ条件で扱ってください
- 調べたのはコミット a1978a2 の時点のソースコードで、範囲は1章に書いたとおりです。範囲の外の練習ページや medium・high の版にも、意図して入れた脆弱性があります
- DVWA は難易度をブラウザの Cookie で切り替えます。これは練習のための仕組みなので、評価の対象から外しました
- 指摘はソースコードを読んだ結果です。実際の動き方は、ウェブサーバーや PHP の設定によって変わることがあります
- 差分の例は、PHP と JavaScript の構文のチェックだけを行っています。適用するときは、テスト環境で動作を確認してください
- 攻撃の手順や、攻撃に使える入力の例は載せていません。鍵とパスワードの値は伏せ字にしています

付録　調査したファイルの一覧

区分	ファイル	行数
共通部分	`about.php`	56
共通部分	`index.php`	47
共通部分	`instructions.php`	67
共通部分	`login.php`	144
共通部分	`logout.php`	17
共通部分	`phpinfo.php`	10
共通部分	`security.php`	89
共通部分	`setup.php`	171
共通部分	`dvwa/includes/dvwaPage.inc.php`	691
共通部分	`dvwa/includes/DBMS/MySQL.php`	142
共通部分	`dvwa/js/dvwaPage.js`	45
共通部分	`dvwa/js/add_event_listeners.js`	24
共通部分	`config/config.inc.php.dist`	56
共通部分	`php.ini`	5
共通部分	`compose.yml`	39
共通部分	`Dockerfile`	25
練習ページの low 版	`vulnerabilities/brute/index.php`	67
練習ページの low 版	`vulnerabilities/brute/source/low.php`	32
練習ページの low 版	`vulnerabilities/sqli/index.php`	82
練習ページの low 版	`vulnerabilities/sqli/source/low.php`	56
練習ページの low 版	`vulnerabilities/xss_s/index.php`	87
練習ページの low 版	`vulnerabilities/xss_s/source/low.php`	22
練習ページの low 版	`vulnerabilities/xss_r/index.php`	66
練習ページの low 版	`vulnerabilities/xss_r/source/low.php`	11
練習ページの low 版	`vulnerabilities/upload/index.php`	75
練習ページの low 版	`vulnerabilities/upload/source/low.php`	19
練習ページの low 版	`vulnerabilities/authbypass/index.php`	77

区分	ファイル	行数
練習ページの low 版	`vulnerabilities/authbypass/source/low.php`	11
練習ページの low 版	`vulnerabilities/authbypass/get_user_data.php`	42
練習ページの low 版	`vulnerabilities/authbypass/change_user_details.php`	52
練習ページの low 版	`vulnerabilities/authbypass/authbypass.js`	53
API の認証部分	`vulnerabilities/api/src/Token.php`	62
API の認証部分	`vulnerabilities/api/src/Login.php`	48
API の認証部分	`vulnerabilities/api/src/LoginController.php`	282
合計	34ファイル	2,772

修正方法の手本として読んだファイル

調査範囲には含めていません。

ファイル	行数
`vulnerabilities/brute/source/impossible.php`	102
`vulnerabilities/sqli/source/impossible.php`	65
`vulnerabilities/xss_s/source/impossible.php`	31
`vulnerabilities/xss_r/source/impossible.php`	18
`vulnerabilities/upload/source/impossible.php`	65
`vulnerabilities/authbypass/source/impossible.php`	13
合計 6ファイル	294