计划了好久终于开坑了

暴力破解

low

源码

```php
<?php

if( isset( $_GET[ 'Login' ] ) ) {
    // Get username
    $user = $_GET[ 'username' ];

    // Get password
    $pass = $_GET[ 'password' ];
    $pass = md5( $pass );

    // Check the database
    $query  = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass';";
    //若or前面语句执行成功则不执行or后面语句 不成功 则判断 $GLOBALS["___mysqli_ston"] 是否为对象 如果是 则mysqli_error() 函数返回最近调用函数的最后一个错误描述。
    //否则mysqli_connect_error() 函数返回上一次连接错误的错误描述。
    $result = mysqli_query($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>' );
    //mysqli_num_rows() 函数返回结果集中行的数量。
    if( $result && mysqli_num_rows( $result ) == 1 ) {
        // Get users details mysqli_fetch_assoc 以数组啷方式返回结果啷一行
        $row    = mysqli_fetch_assoc( $result );
        $avatar = $row["avatar"];

        // Login successful
        $html .= "<p>Welcome to the password protected area {$user}</p>";
        $html .= "<img src=\"{$avatar}\" />";
    }
    else {
        // Login failed
        $html .= "<pre><br />Username and/or password incorrect.</pre>";
    }
    #关闭mysql连接 $GLOBALS["___mysqli_ston"] 调试着是 mysqli 还没找到 这个东东是在哪初始化啷
    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
}

//总之分析下来就是没有次数啷限制 可以进行暴力破解 username 也没有做任何啷sql注入防护 pass参数有md5 基本杜绝了sql注入的可能性
?>
```

medium

```php
$user = $_GET[ 'username' ];
$user = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],

// Sanitise password input
```

相比Low级别的代码，Medium级别的代码主要增加了mysql_real_escape_string函数，这个函数会对字符串中的特殊符号（x00，n，r，，’，”，x1a）进行转义，基本上能够抵御sql注入攻击，说基本上是因为查到说 MySQL5.5.37以下版本如果设置编码为GBK，能够构造编码绕过mysql_real_escape_string 对单引号的转义（应该就是宽字节注入）；然后登陆失败加了一个sleep(2)，只要修改下爆破脚本就可以绕过了。

high

```php
checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

// Sanitise username input
$user = $_GET[ 'username' ];
$user = stripslashes( $user );
```

每次服务器返回的登陆页面中都会包含一个随机的user_token的值，用户每次登录时都要将user_token一起提交。服务器收到请求后，会优先做token

的检查，再进行sql查询。但是可以写py脚本，先获取登陆页面嗝返回嗝token，然后再发送登陆请求

同时使用了stripslashes（去除字符串中的反斜线字符,如果有两个连续的反斜线,则只去掉一个）进一步防范了sql注入

impossable

增加了错误次数验证 那就没办法咯

```
// Default values
$total_failed_login = 3;
$lockout_time       = 15;
$account_locked     = false;
```

代码执行

low

没有任何过滤 直接 && ; 隔开 127.0.0.1 && whoami

index.php low.php medium.php

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command. 先判断系统
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    $html .= "<pre>{$cmd}</pre>";
}

?>
```

medium

过滤了&&；

```
// Set blacklist
$substitutions = array(
    '&&' => '',
    ';'  => '',
);

// Remove any of the charactars in the array (blacklist).array_keys 返回keys数组 然后循环替换
$target = str_replace( array_keys( $substitutions ), $substitutions, $target );
```

只过滤了上述两种字符，这里我们还可以用&号、管道符|等直接绕过

impossible 绕不过去233

```php
checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

// Get input
$target = $_REQUEST[ 'ip' ];
$target = stripslashes( $target ); //删除反斜杠

// Split the IP into 4 octects
$octet = explode( ".", $target ); //打散为数组

// Check IF each octet is an integer #判断每个是否为数字 然后重新组装
if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof( $octet ) == 4 ) ) {
    // If all 4 octets are int's put the IP back together.
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];
```

下面是CSRF

low映入眼帘一点防护都没做 pass_new还存在sql注入入

```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Get input
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Do the passwords match?
    if( $pass_new == $pass_conf ) {
        // They do!
        $pass_new = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((trigger_
        $pass_new = md5( $pass_new );

        // Update the database
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . dvwaCurrentUser() . "';";
        $result = mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($__

        // Feedback for the user
        $html .= "<pre>Password Changed.</pre>";
    }
    else {
        // Issue with passwords matching
        $html .= "<pre>Passwords did not match.</pre>";
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
}
```

medium 增加了refer字段嘚检测

```php
// Checks to see where the request came from stripos() 函数查找字符串在另一字符串中第一次出现的位置
if( stripos( $_SERVER[ 'HTTP_REFERER' ] ,$_SERVER[ 'SERVER_NAME' ]) !== false ) {
    // Get input
```

high 增加了token检测

因为token是写在页面里嘚 如果没有xss 有个毛线用啊 有了xss 要csrf有个毛用

impossible 直接提示输入旧密码 简单省事咯

文件包含

low 真香

low.php

```php
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

?>
```

medium 首先使用str_replace函数是极其不安全的，因为可以使用双写绕过替换规则其次我们还有file ftp协议呢 或者伪协议

```php
// Input validation
$file = str_replace( array( "http://", "https://" ), "", $file );
$file = str_replace( array( "../", "..\"" ), "", $file );

?>
```

high file协议

```php
// Input validation //必须以file开头
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
```

impossible 直接白名单锁死

文件上传

low well没有任何过滤

```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( $_FILES[ 'uploaded' ][ 'name' ] );//$_FILES[ 'uploaded' ][ 'name' ] 上传文件名称 basename 获取文件名

    //  move_uploaded_file 移动文件 成功返回true 第一个参数是要移动哪文件
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
        // No
        $html .= '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        $html .= "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}
```

medium 制作了文件类型嗝过滤 burp改下包就成

```php
// Is it an image?
if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
    ( $uploaded_size < 100000 ) ) {
```

high 这次对文件后缀做了检测 这个怎么用00截断啊... 网上都说可以....

```php
// File information
$uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
$uploaded_ext  = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1);//获取文件后缀名
$uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
$uploaded_tmp  = $_FILES[ 'uploaded' ][ 'tmp_name' ];

// Is it an image?
if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower( $uploaded_ext ) == "jpeg" || strtolower( $uploaded_ext ) == "png" ) &&
    ( $uploaded_size < 100000 ) &&
    getimagesize( $uploaded_tmp ) ) {
```

sql注入

low 没有做任何嘚防护 直接拼接的sql语句

medium 对id函数增加了 mysqli_real_escape_string过滤 \x00,\n,\r,\,',",\x1a进行转义 但是查询字段不是字符型 不需要单引号来闭合 所以无效 就算是字符型 如果为gbk编码 也坑会用宽字节绕过

high 与low相比 值增加了一个limit 1 emmm...敢问这个有啥子用嚒233

impossable

```php
if(is_numeric( $id )) { //检测是否为数字
    // Check the database 这个应该就是预处理吧
    $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
    $data->bindParam( ':id', $id, PDO::PARAM_INT );
    $data->execute();
    $row = $data->fetch();

    // Make sure only 1 result is returned
    if( $data->rowCount() == 1 ) {
        // Get values
        $first = $row[ 'first_name' ];
        $last  = $row[ 'last_name' ];

        // Feedback for end user
        $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }
```

xss

真是懒得分析了233

preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET['name'])

还不如搜一下这个函数