Minor Project-"CS April 20 - May 20”

CS-**AprCS04B4**

VERZEO
MINOR PROJECT

Submitted by Antara Sinha

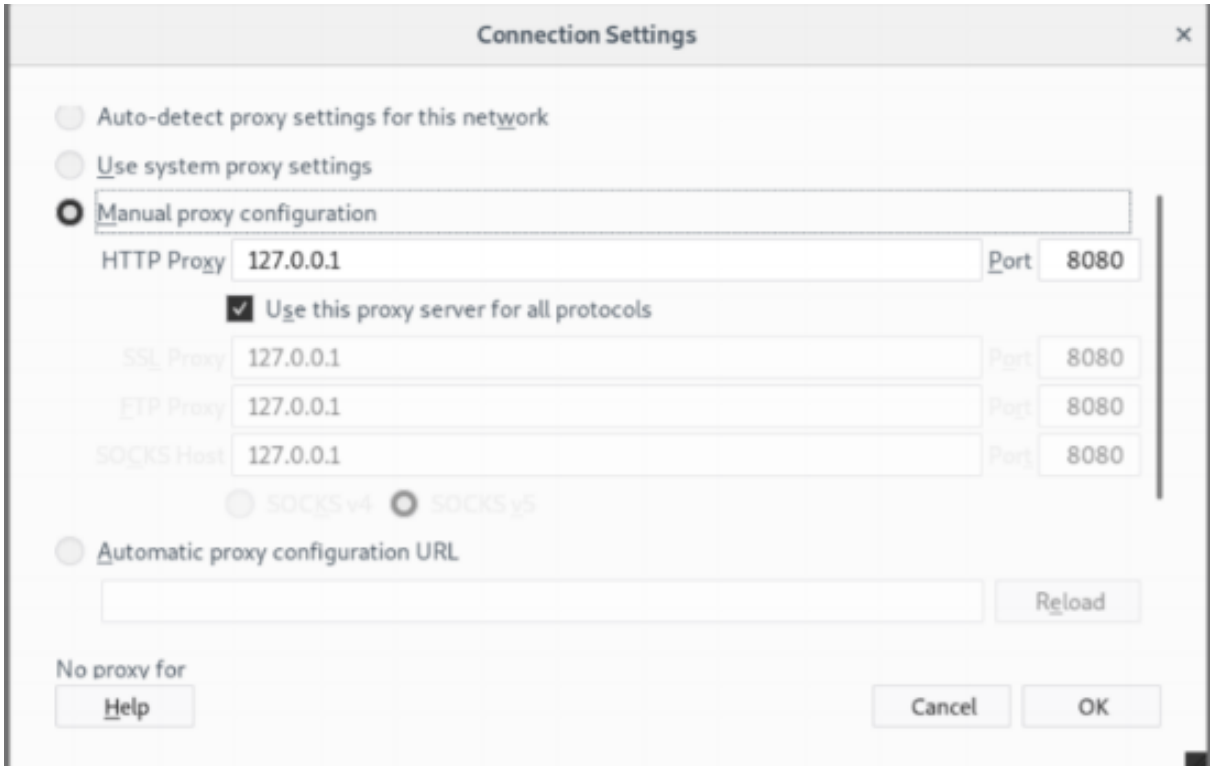
Email- antarasinha00@gmail.com

Date 26/05/20

For this minor project I was provided with a website http://lab.hackerinside.xyz/login.php and was asked to test this website for vulnerabilities and find flags and suggest ways to patch this bug.

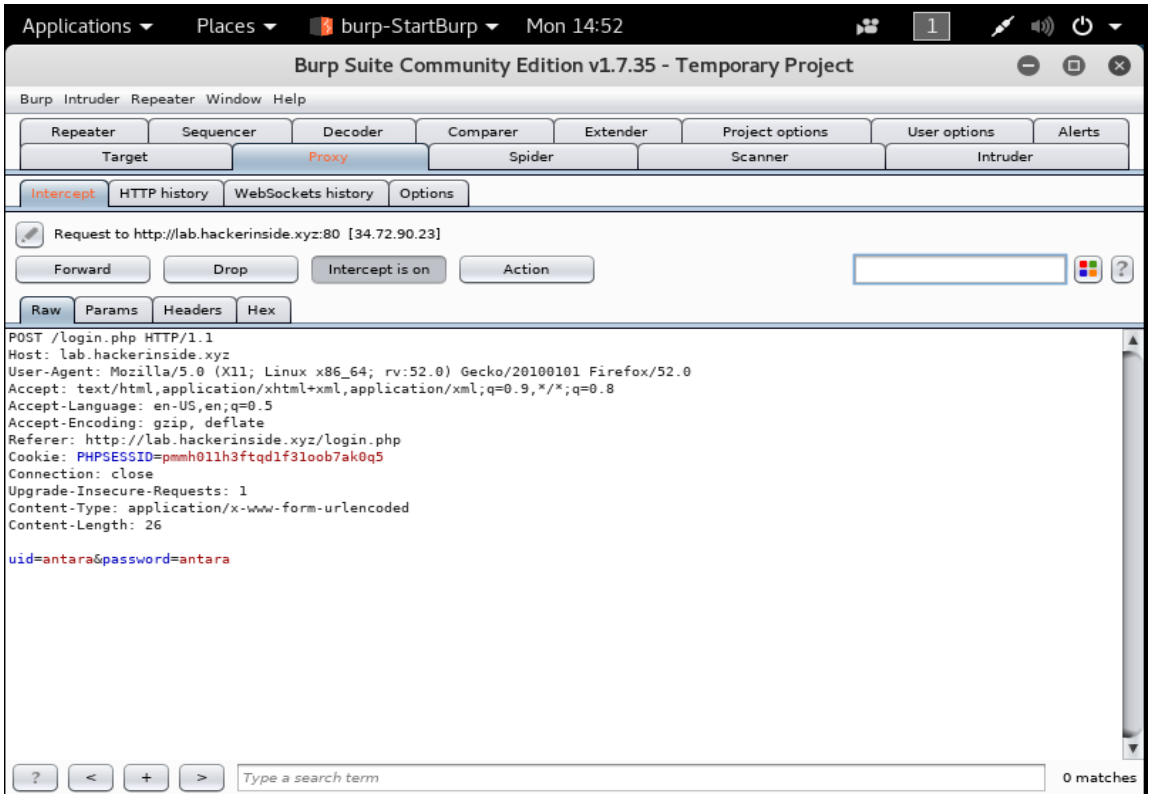
VULNERABILITY DETCTION AND EXPLOITATION

- After **vulnerability scanning** I found multiple vulnerabilities and decided to the SQL injection vulnerability.
- In order to exploit the **SQL injection** vulnerability I first configured the network proxy setting on Firefox browser so that Burp Suite can intercept the requests. So I set it to manual proxy setting of Firefox .

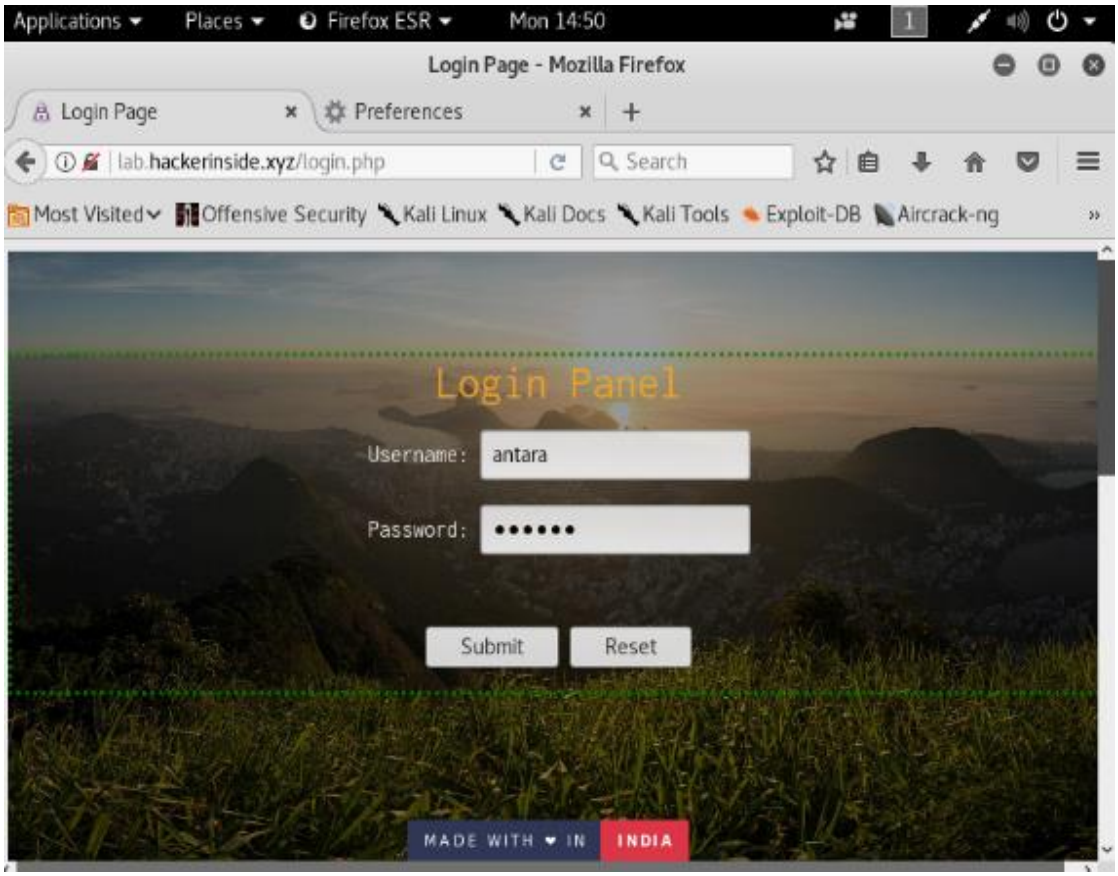


(MANUAL PROXY SETTING OF FIREFOX)

- Then I start burp suit and make sure that the intercept mode is on and login to the website to capture all parameters and fields.



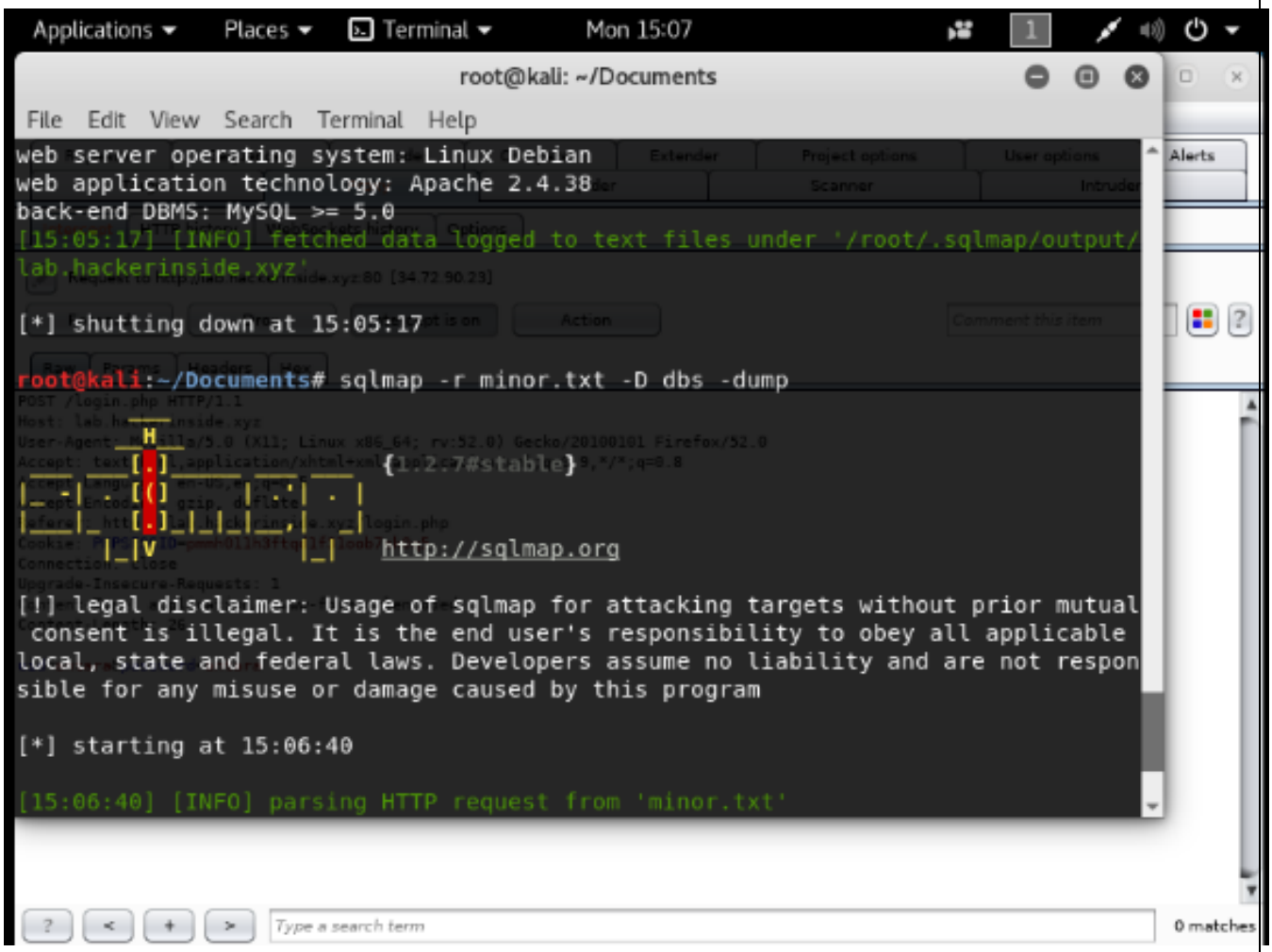
(BURP SUITE INTERCEPT PAGE)



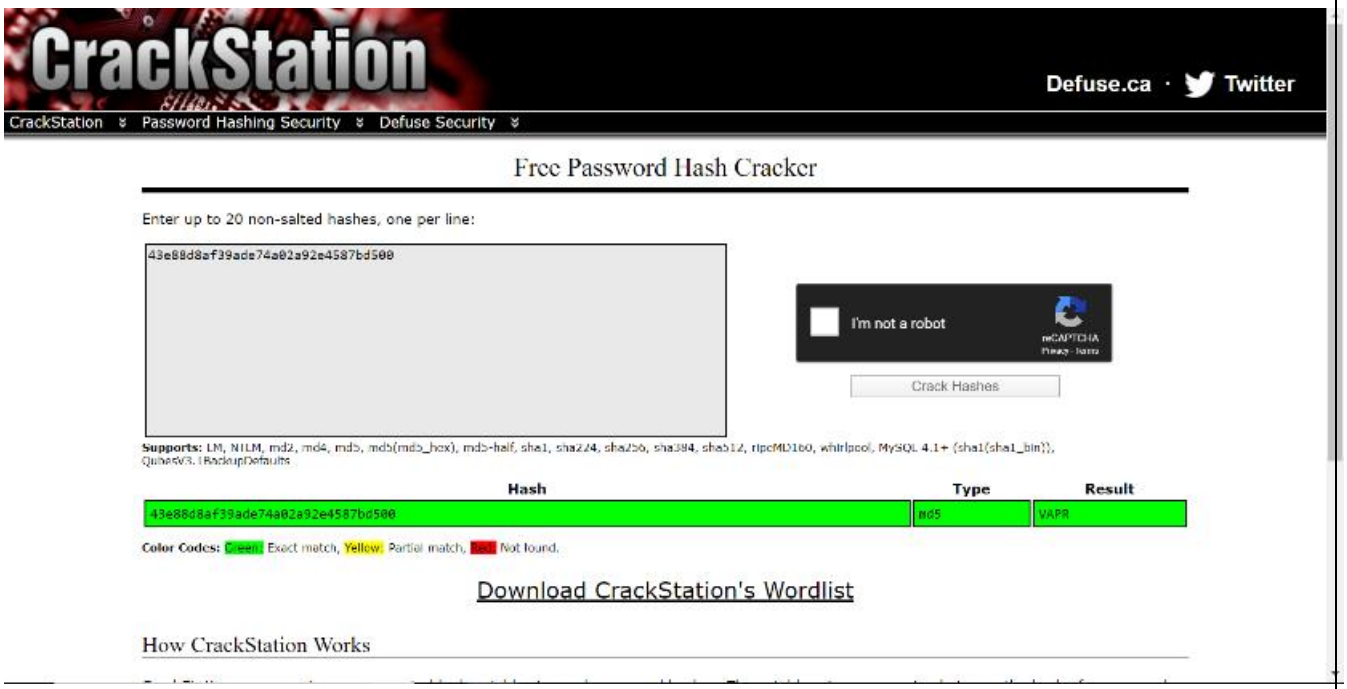
(http://lab.hackerinside.xyz/login.php LOGIN PAGE)

GRAB THE FLAG

- Then I saved the file as minor.txt file and opened the terminal to locate it and then parse it to SQLMAP tool to find the field and database vulnerable to SQL injection. After doing this we found database 'dbs'.
- Then using "sqlmap –r minor.txt -D dbs --dump" I dumped all the tables along with thw values
- In the table flag of dbs there was an entry so the flag column has a hash value which is MD5 hash and the value is –' **VAPR**'.



(FLAG VALUE)



(USING CRACKSTATION TO TO GET THE RESULT FOR FRO THE HASH WHICH IS OF TYPE MD5)

PATCH FOR THIS BUG

SQL Injection attacks are unfortunately very common, and this is due to two factors:

- the significant prevalence of SQL Injection vulnerabilities, and
- the attractiveness of the target (i.e., the database typically contains all the interesting/critical data for your application).

To fix this bug we could use Prepared statements and Php data objects

Using Prepared Statements

All developers should use prepared statements with variable binding (parameterised queries) . Here the developer should first define all the SQL code, and then pass each parameter to the query.

This would help distinguish code and data regardless of what is being entered ,therefore even if the attacker did make changes it would not change the intent of the query ,for example if an attacker were to enter the user ID of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'='1.

PDO

PHP Data Objects (using binPharma()) provide methods using which we can make parameterised queries easily, and it works with multiple databases .

The given code suggest changes that can be made

- According to db_config.php available on https://github.com/anir0y/ProjectZero/blob/master/db_config.php . The code can be changed as :

```
<?php
// init
$con = new PDO("localhost", "gh0st", "gh0st", "dbs");
if ($con->connect_error){
 die("connection failed: ".$con->connect_error);
}
```

- According to login.php available on https://github.com/anir0y/ProjectZero/blob/master/login.php. The code can be changed as:

```
<?php
ob_start();
session_start();
include("db_config.php");
ini_set('display_errors', 1);
?>
<html lang="en">
<head>
    <link href="css/htmlstyles.css" rel="stylesheet">
    <link rel=stylesheet href="https://fortawesome.github.io/FontAwesome/assets/font-awesome/css/font-awesome.css">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Login Page</title>
</head>
```

```
<body>
<center>
<section class="header">
      <div class="mask">
            <div class="text-align">
                  <div class="loginheader">
                  <h1>Login Panel</h1>

     <form method="POST" autocomplete="off">
            <p style="color:white">
                  Username: <input type="text" id="uid"
placeholder="username" name="uid"><br /></br />
                  Password: <input type="password"
id="pass"placeholder="Password" name="password">
                        </p>
                        <br />
                        <p>
                        <input type="submit" value="Submit"/>
                        <input type="reset" value="Reset"/>
                        </p></div>
                  </form></center>
            </div>
            <a href="#section" class="section"><i class="fa fa-chevron-circle-down"></i></a>
          </div>
</section>
            </center>
 </div>
            <br />
      <div class="row marketing">
      <div class="col-lg-6">
<?php
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
if (!empty($_GET['msg'])) {
            echo "<center font style=\"color:#FF0000\">Ohh!Wrong Creds :(
```

```
.<br\></font\>";
}
//echo md5("pa55w0rd");

if (!empty($_REQUEST['uid'])) {
$username = ($_REQUEST['uid']);
$pass = $_REQUEST['password'];

$q = "SELECT * FROM users where username='":username."' AND password =
'".md5(:pass)."" ;

$sth->$dbh->prepare($q);
$sth->bindParam(':username',$username,':pass',$pass);
//echo $q;
    if (!mysqli_query($con,$q))
    {
        die('Error: ' . mysqli_error($con));
    }
    $result = mysqli_query($con,$q);
    $row_cnt = mysqli_num_rows($result);

    if ($row_cnt > 0) {
    $row = mysqli_fetch_array($result);
    if ($row){
    //$_SESSION["id"] = $row[0];
    $_SESSION["username"] = $row[1];
    $_SESSION["name"] = $row[3];
    //ob_clean();
    header('Location:home.php');
    }
}
    else{
        echo "<center><font style=\"color:#FF0000\"><br \>Invalid
password! <b>Try Harder<b> </font\>";
```

```
            }
    }
    //}
    ?>
                </div>
                </div>
                </div>
    <section class="section-one" id="section">
        <div class="container">
                <div class="row">
                        <div class="col-md-12 text-center">
                                <center> <h1>About this project</h1>
    <p>Project Zero is a simple vulnerable web-application, created for the
    people who want to learn about CyberSecurity. </p><em>this simple box is very
    easy to PWN and perfect for classroom training.</em></p>
     Our target is to use this module to teach school kids/college studends about
    web application vulnerablilyty. </p>
        <br></br>
    </i style="color:white">More attack modules will be added soon! As this one is
    developing by my students it will be slow, if you want to contribute for this
    Project feel free to send a pull request on github or DM me on twitter <a
    href="https://twitter.com/anir0y">@anir0y<a></i></center>


                        </div>
                </div>
        </div>
    </section>
    </body>
    </html>
    <?php include("footer.php"); ?>
```

Therefore when we think of prepared strings we assemble the string with placeholders for the data to be inserted and apply the data in the same sequence as the placeholders.so in SQL prepared statements we store a prepared statement and feed data into it and sanitize it upon execution. In order to add more security we could do a few more things because the only way

to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements.

The developer must sanitize all inputs and not only web form inputs such as login forms also the application code should never use the input directly, all potential malicious code elements single quotes should be removed. Along with that it is good practice to turn off the visibility of database errors as they can be exploited to gain information about the data base.
