# Brute Force

Brute Force，即暴力（破解），是指黑客利用密码字典，使用穷举法猜解出用户口令，是现在最为广泛使用的攻击手法之一，如2014年轰动全国的12306“撞库”事件，实质就是暴力破解攻击。

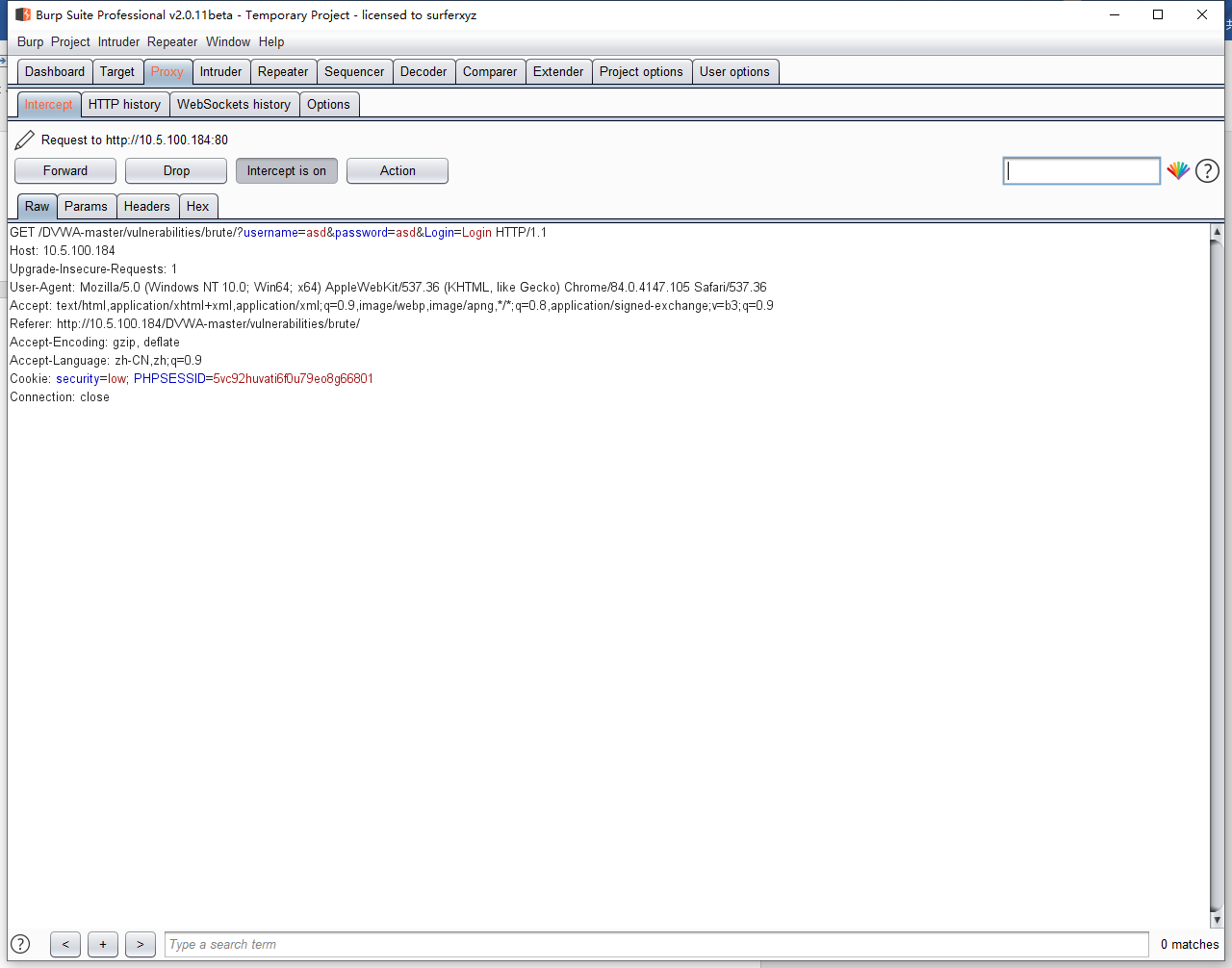
# Level:Low

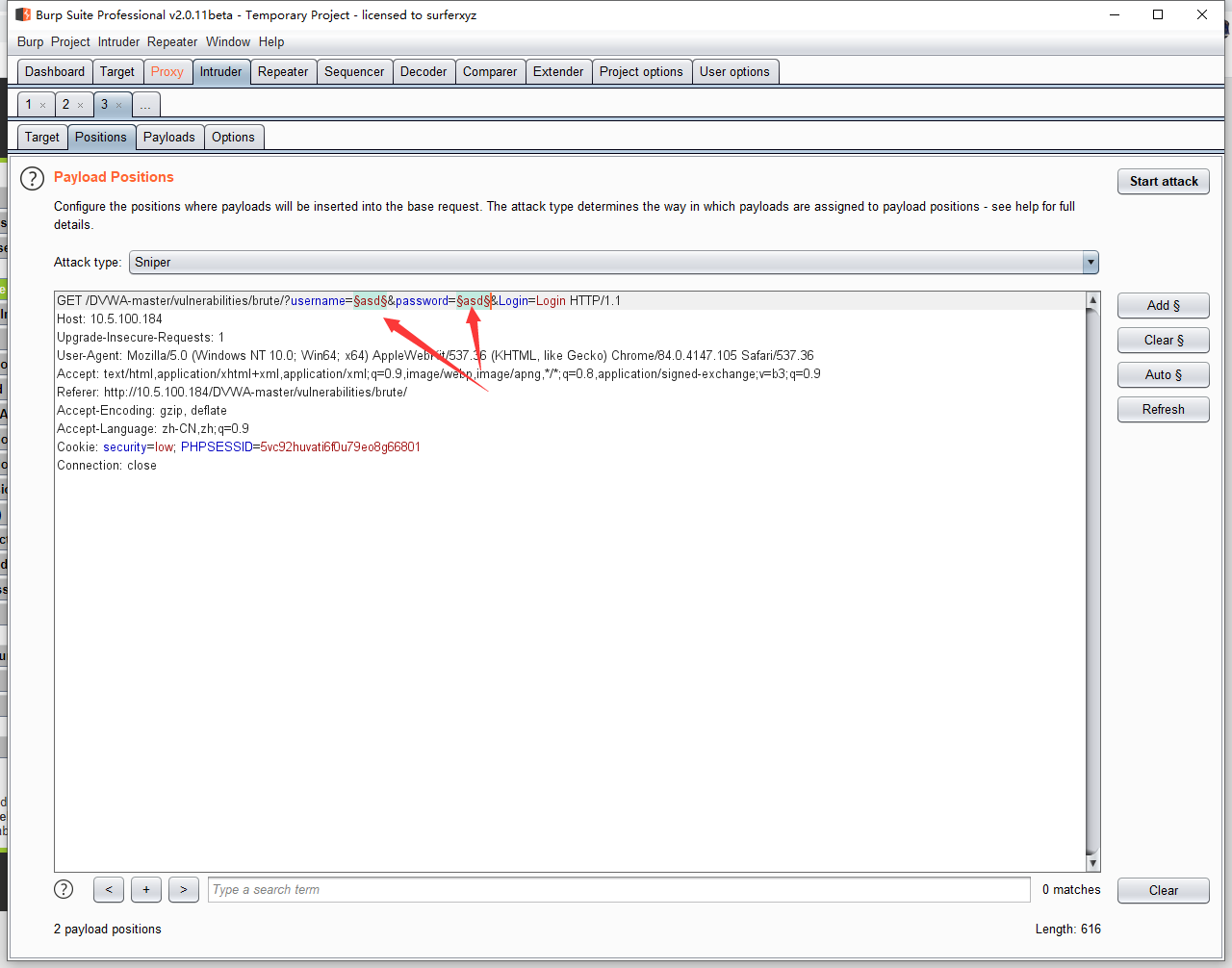


Low等级的代码只是验证了login是否被设置，没有任何防爆破机制，且对参数username、password没有做任何过滤，存在明显的sql注入漏洞

爆破工具使用BP就行了

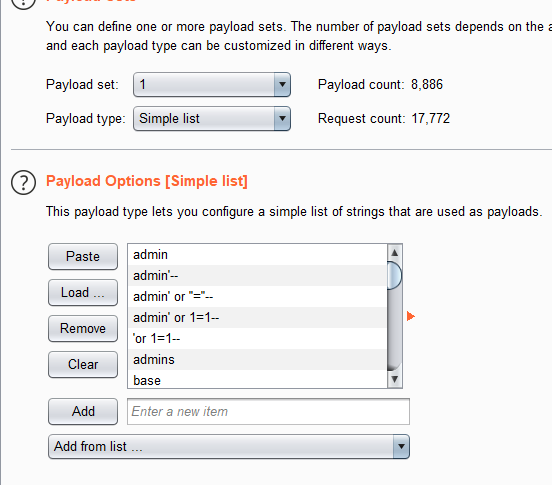
首先抓包



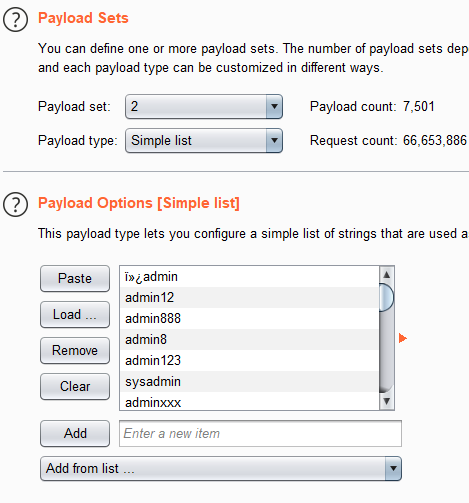


给两个值弄变量

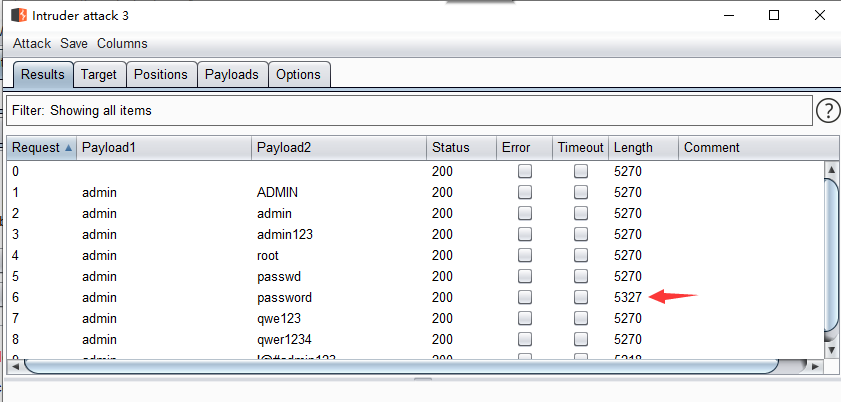
给第一个变量选入字典



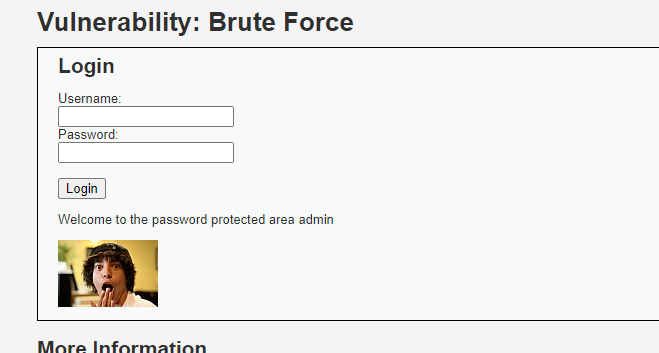
给第二个变量选入字典



然后进行爆破



爆破出用户名admin和密码password



登录成功

# Level:Medium

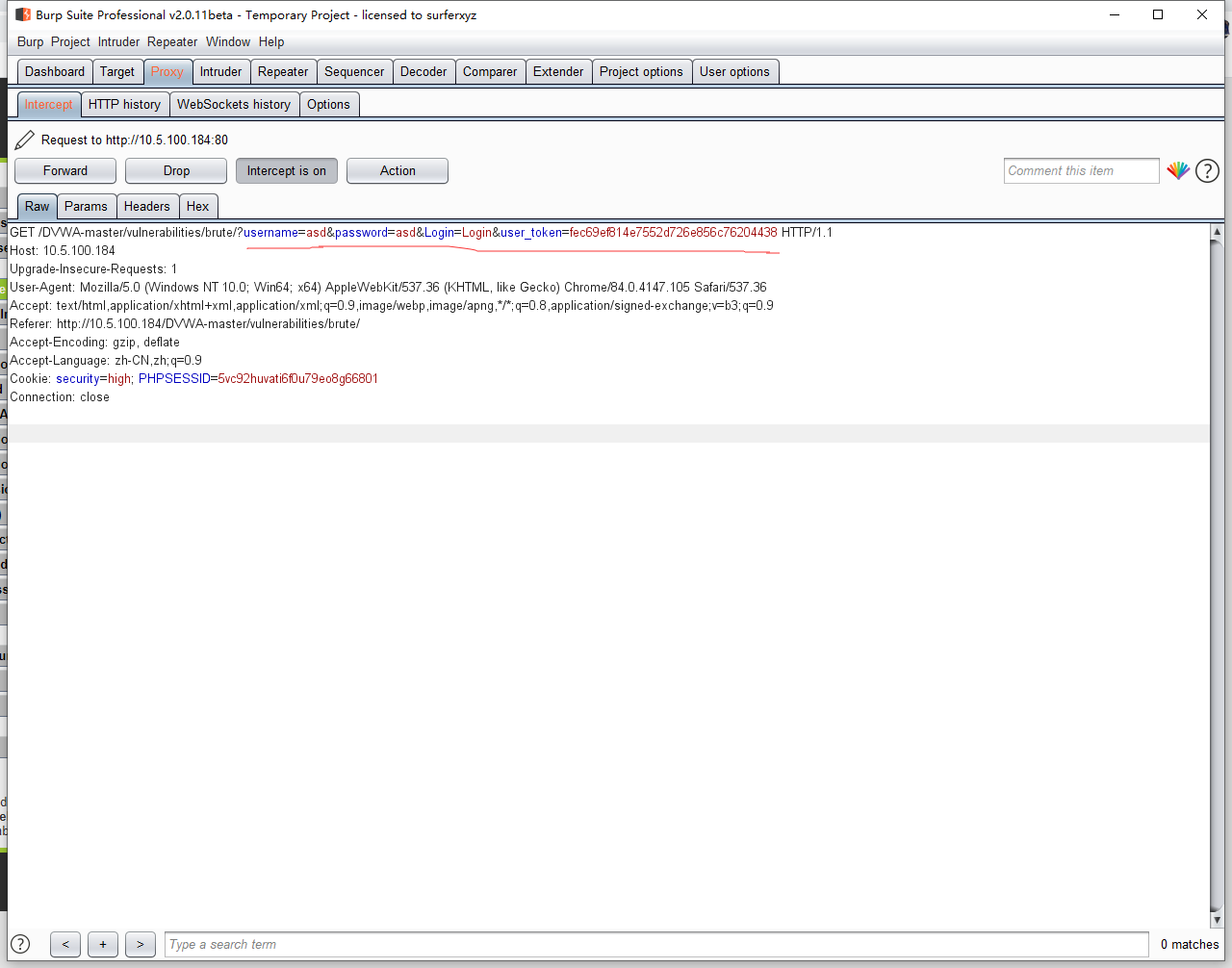


Medium等级的源码Medium级别的代码主要增加了mysql\_real\_escape\_string函数，这个函数会对字符串中的特殊符号（x00，n，r，，'，"，x1a）进行转义，基本上能够抵御sql注入攻击，虽然sql注入不再有效，但依然可以使用Burpsuite进行爆破，与Low级别的爆破方法基本一样，就不掩饰了

# Level:High

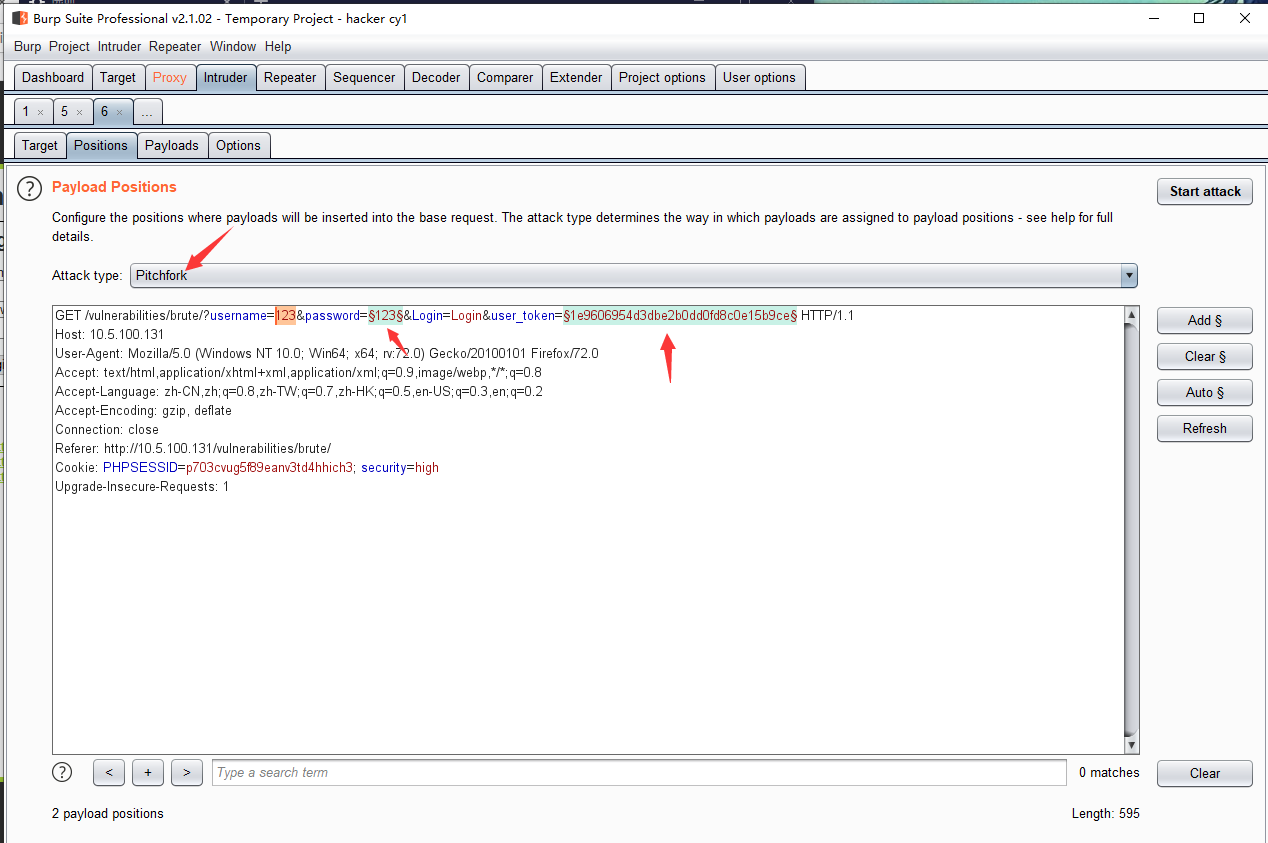


High级别的代码加入了Token，可以抵御CSRF攻击，同时也增加了爆破的难度，通过抓包，可以看到，登录验证时提交了四个参数：username、password、Login以及user\_token。

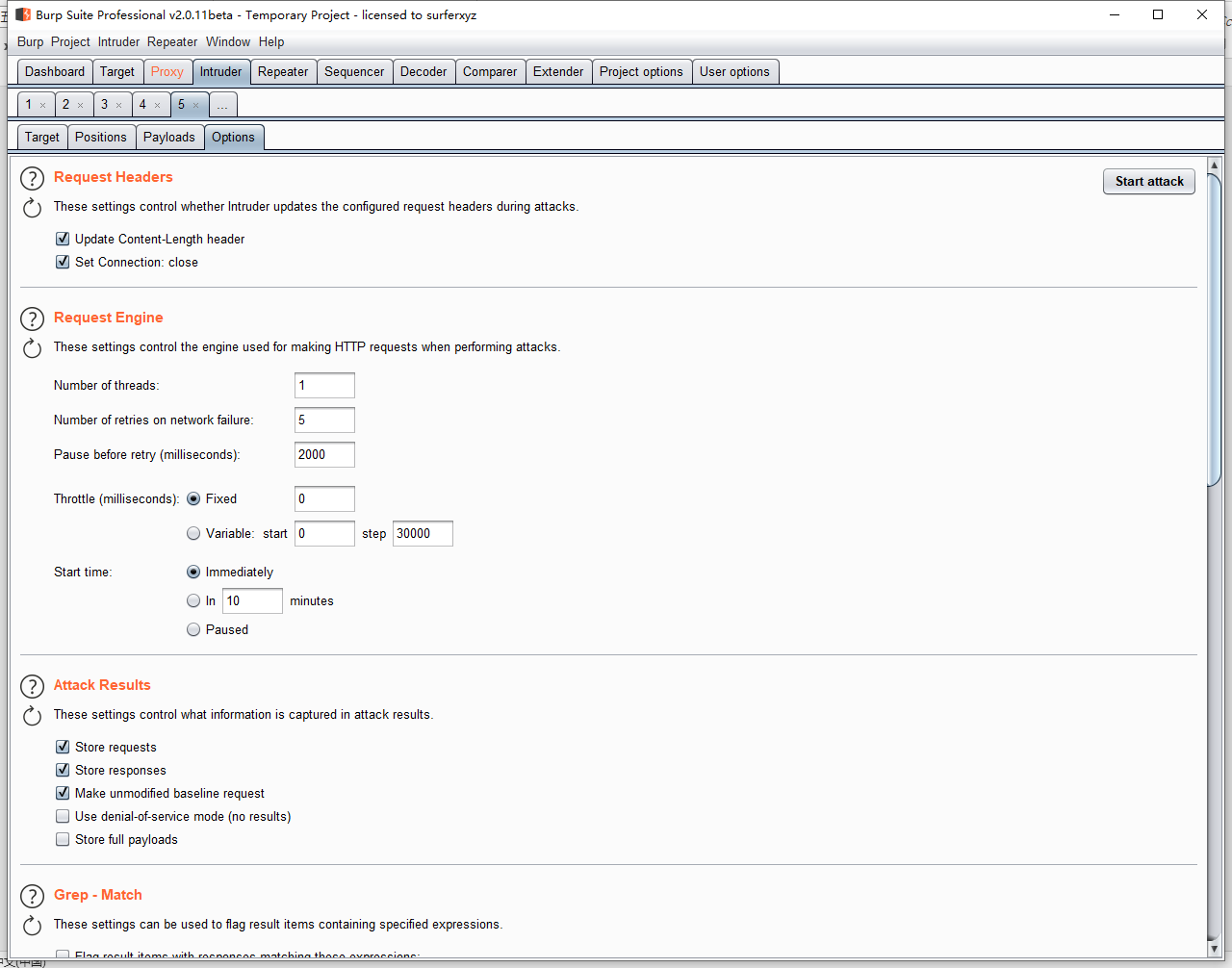


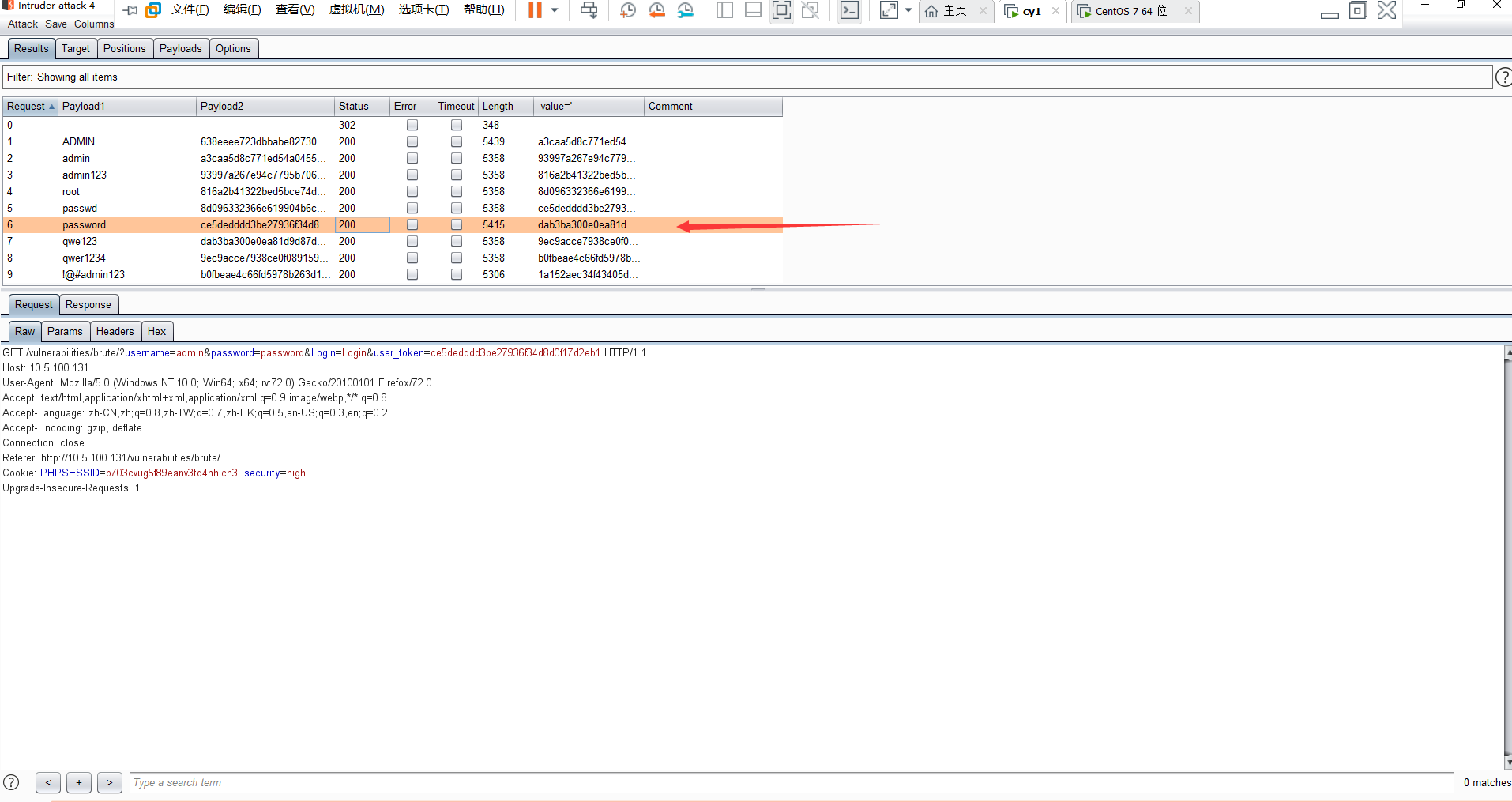
同时，High级别的代码中，使用了stripslashes（去除字符串中的反斜线字符,如果有两个连续的反斜线,则只去掉一个）、 mysql\_real\_escape\_string对参数username、password进行过滤、转义，进一步抵御sql注入。

设置两个参数 password和user\_token为变量，攻击类型选择pitchfork，意思是草叉模式（Pitchfork ）——它可以使用多组Payload集合，在每一个不同的Payload标志位置上（最多20个），遍历所有的Payload。举例来说，如果有两个Payload标志位置，第一个Payload值为A和B，第二个Payload值为C和D，则发起攻击时，将共发起两次攻击，第一次使用的Payload分别为A和C，第二次使用的Payload分别为B和D



设置参数，在option选项卡中将攻击线程thread设置为1，因为Recursive\_Grep模式不支持多线程攻击，然后选择Grep-Extract，意思是用于提取响应消息中的有用信息，点击Add，如下图进行设置，最后将Redirections设置为Always





得到密码

# Level:Impossible

<?php  
  
if( isset( $\_POST[ 'Login' ] ) && isset ($\_POST['username']) && isset ($\_POST['password']) ) {  
    // Check Anti-CSRF token  
    checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );  
  
    // Sanitise username input  
    $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)) ? "" : ""));  
  
    // Sanitise password input  
    $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 );  
  
    // Default values  
    $total\_failed\_login = 3;  
    $lockout\_time       = 15;  
    $account\_locked     = false;  
  
    // Check the database (Check user information)  
    $data = $db->prepare( 'SELECT failed\_login, last\_login FROM users WHERE user = (:user) LIMIT 1;' );  
    $data->bindParam( ':user', $user, PDO::PARAM\_STR );  
    $data->execute();  
    $row = $data->fetch();  
  
    // Check to see if the user has been locked out.  
    if( ( $data->rowCount() == 1 ) && ( $row[ 'failed\_login' ] >= $total\_failed\_login ) )  {  
        // User locked out.  Note, using this method would allow for user enumeration!  
        //echo "<pre><br />This account has been locked due to too many incorrect logins.</pre>";  
  
        // Calculate when the user would be allowed to login again  
        $last\_login = strtotime( $row[ 'last\_login' ] );  
        $timeout    = $last\_login + ($lockout\_time \* 60);  
        $timenow    = time();  
  
        /\*  
        print "The last login was: " . date ("h:i:s", $last\_login) . "<br />";  
        print "The timenow is: " . date ("h:i:s", $timenow) . "<br />";  
        print "The timeout is: " . date ("h:i:s", $timeout) . "<br />";  
        \*/  
  
        // Check to see if enough time has passed, if it hasn't locked the account  
        if( $timenow < $timeout ) {  
            $account\_locked = true;  
            // print "The account is locked<br />";  
        }  
    }  
  
    // Check the database (if username matches the password)  
    $data = $db->prepare( 'SELECT \* FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );  
    $data->bindParam( ':user', $user, PDO::PARAM\_STR);  
    $data->bindParam( ':password', $pass, PDO::PARAM\_STR );  
    $data->execute();  
    $row = $data->fetch();  
  
    // If its a valid login...  
    if( ( $data->rowCount() == 1 ) && ( $account\_locked == false ) ) {  
        // Get users details  
        $avatar       = $row[ 'avatar' ];  
        $failed\_login = $row[ 'failed\_login' ];  
        $last\_login   = $row[ 'last\_login' ];  
  
        // Login successful  
        echo "<p>Welcome to the password protected area <em>{$user}</em></p>";  
        echo "<img src=\"{$avatar}\" />";  
  
        // Had the account been locked out since last login?  
        if( $failed\_login >= $total\_failed\_login ) {  
            echo "<p><em>Warning</em>: Someone might of been brute forcing your account.</p>";  
            echo "<p>Number of login attempts: <em>{$failed\_login}</em>.<br />Last login attempt was at: <em>${last\_login}</em>.</p>";  
        }  
  
        // Reset bad login count  
        $data = $db->prepare( 'UPDATE users SET failed\_login = "0" WHERE user = (:user) LIMIT 1;' );  
        $data->bindParam( ':user', $user, PDO::PARAM\_STR );  
        $data->execute();  
    } else {  
        // Login failed  
        sleep( rand( 2, 4 ) );  
  
        // Give the user some feedback  
        echo "<pre><br />Username and/or password incorrect.<br /><br/>Alternative, the account has been locked because of too many failed logins.<br />If this is the case, <em>please try again in {$lockout\_time} minutes</em>.</pre>";  
  
        // Update bad login count  
        $data = $db->prepare( 'UPDATE users SET failed\_login = (failed\_login + 1) WHERE user = (:user) LIMIT 1;' );  
        $data->bindParam( ':user', $user, PDO::PARAM\_STR );  
        $data->execute();  
    }  
  
    // Set the last login time  
    $data = $db->prepare( 'UPDATE users SET last\_login = now() WHERE user = (:user) LIMIT 1;' );  
    $data->bindParam( ':user', $user, PDO::PARAM\_STR );  
    $data->execute();  
}  
  
// Generate Anti-CSRF token  
generateSessionToken();  
  
?>

可以看到Impossible级别的代码加入了可靠的防爆破机制，当检测到频繁的错误登录后，系统会将账户锁定，爆破也就无法继续，同时采用了更为安全的PDO（PHP Data Object）机制防御sql注入，这是因为不能使用PDO扩展本身执行任何数据库操作，而sql注入的关键就是通过破坏sql语句结构执行恶意的sql命令。