Check Point Report

Vulnerable Application: BodgeIt

Team Name: AA Team

Kevin Wu 0808148w

Claudia Ortiz-Duron 2412650o

Laimonas Samalius 244831s

Sultan Altwijri 2312914a

Ali Rashed A Al Qarni 2286368a

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerability | Type of Vulnerability | Details of the Attack | Screenshot |
| Login as test@thebodgeitstore.com | SQL Injection (Injection flaw) | Login using test@thebodgeitstore.com' or ''=' was entered in username field | Figure 1 |
| Login as user1@thebodgeitstore.com | SQL Injection (Injection flaw) | ' or ''=' was entered in both id and password to log on | Figure 2 |
| Login as admin@thebodgeitstore.com | SQL Injection (Injection flaw) | Login using admin@thebodgeitstore.com' or ''=' was entered in username field | Figure 3 |
| Find hidden content as a non admin user | Broken Access Control/Failure to restrict URL Access | Exploring URL links is noticeable that they have similar pattern:  http://192.168.56.101/bodgeit/home.jsp;http://192.168.56.101/bodgeit/about.jsp;http://192.168.56.101/bodgeit/contact.jsp  We try to check if Admin page uses the same address pattern. Entering http://192.168.56.3/bodgeit/admin.jsp lets us access unprotected Admin page with hidden information for non admin user. | Figure 4 |
| Find diagnostic data | Security Misconfiguration / Leftover Debug Code | Check on the web site all links by adding at the end of their URL addresses this code ?debug=true. If any debugging data was left by the developer, then the webpage will render it. In this case “DEBUG basketid = 2". | Figure 5 |
| Level 1: Display a popup using: <script>alert("XSS")</script> | Cross Site Scriptin | Go to Search page, enter following code into form field “<script>alert("XSS")</script>” and pop up appeared | Figure 6 |
| Level 2: Display a popup using: <script>alert("XSS")</script> | Cross-Site Scripting | Register a new account and entering aateam@legacy.com<script>alert(“XSS”)</script> in the username box | Figure 7 |
| Level 3: Display a popup using: <script>alert("XSS")</script> | Cross-Site Scripting | Login with aateam@legacy.com<script>alert’ or ‘’=’<script>(“XSS”)</script> in the username. | Figure 8 |
| Access someone else’s basket | Broken Authentication | Log in as an Admin and from Admin page check how basket ID’s are assigned. Then we can to try access someone else basket by manipulating cookies. In this case we amended basked id value in the cookie “b\_id:8” to “b\_id:1”. After pressing “Update basket” we have successfully, accessed another user’s basket. “b\_id:1” | Figure 9 |
| Force someone to add an item to their basket when they visit your webpage. | Cross-Site Request Forgery | Notice that items are added to the basket by a post method. If the attacker views the network data, they can inspect the post request traffic. By clicking on the ‘edit and resent’ tab, the hacker can view the request body. This can then be added at the end of the link, e.g. http://192.168.56.101/bodgeit/basket.jsp?productid=8&price=3.7&quantity=1, which will send a query string to the server for this item to be added to the basket. An attacker might replace a normal link, such as the home link, with the query string link (by inspecting the link element and replacing the ahref attribute to this link). A victim can click the link unaware that they’ve been attacked | Figure 12 |
| Get the store to owe you money | Broken Access Control  / Input validation. | Add purchase item to basket and update basket content. Next, explore the basket web page code trying to find the block for any relevant code related to the quantity of purchased items. After locating this, the value of number of items in basket can be changed to a negative number. Clicking update basket afterwards . The web application processes this as if it was trusted data, making the store owe money to the attacker | Figure 10 |
| Change your password via a GET request | Information exposure through query strings in URL | Click view page source, form submit is set to POST. Inspect one of the password field, replace FORM with GET – click change password. Notice new password is displayed in url bar | Figure 11 |
| 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. |  |  |  |

Screendumps

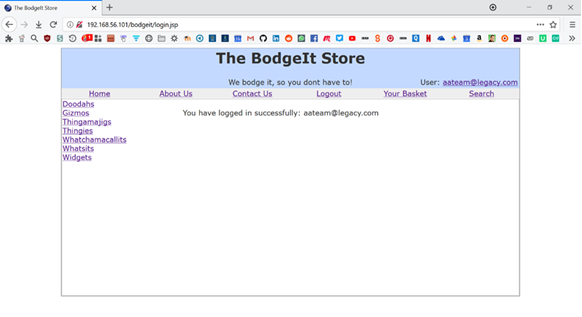
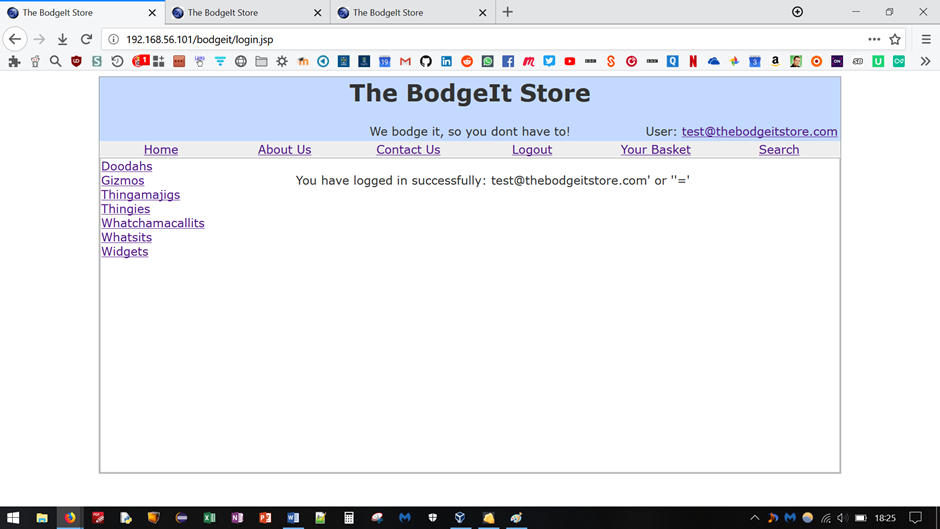
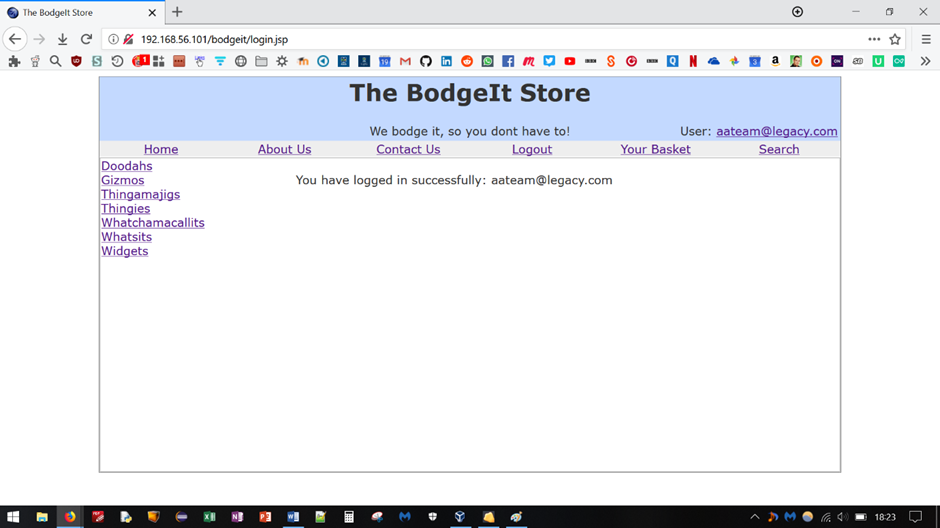


Figure 1: Login as [test@thebodgeitstore.com](mailto:test@thebodgeitstore.com) via a sql query



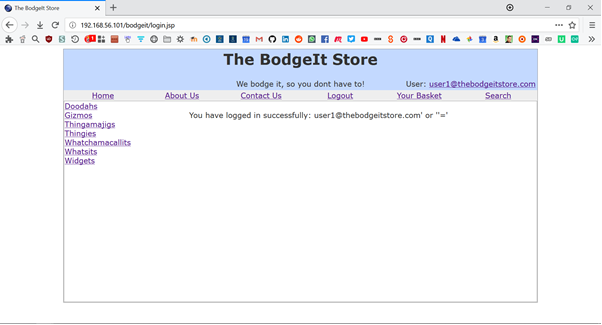
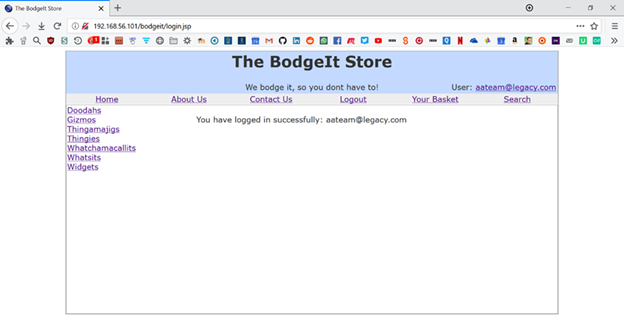


Figure 2: Login as user1@thebodgeitstore.com via SQL query



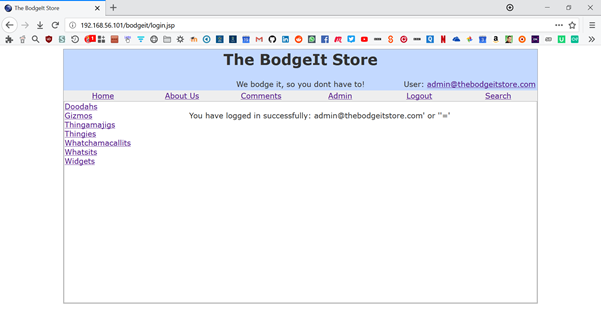


Figure 3: Login as admin@thebodgeitstore.com via a sql injection

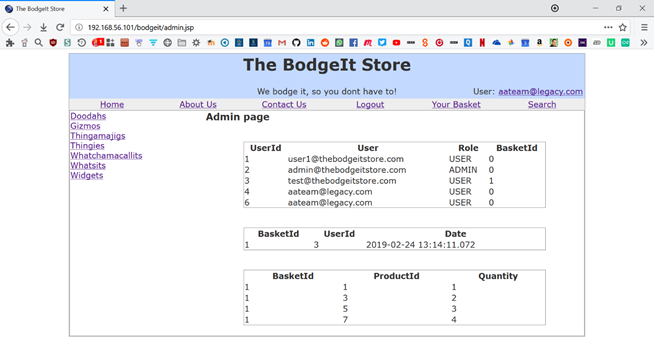


Figure 4: Find hidden content as non-admin user via brute force

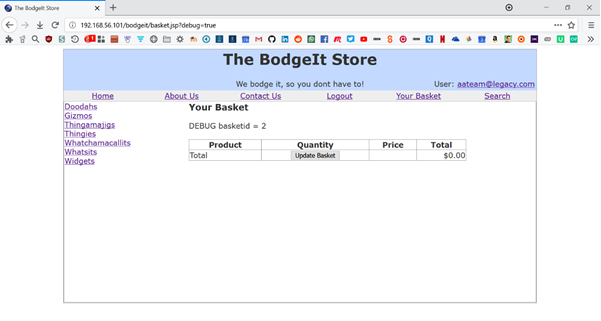


Figure 5: Find diagnostic data

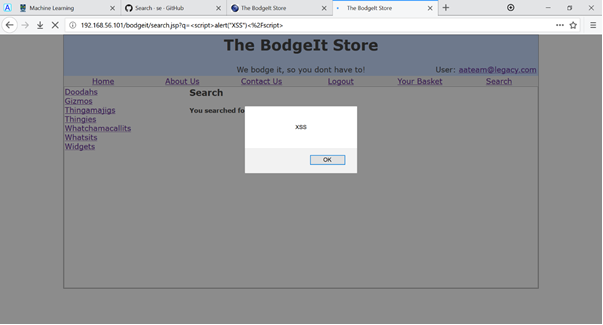


Figure 6: Level 1: Display a popup using: <script>alert("XSS")</script> via SQL query in search box

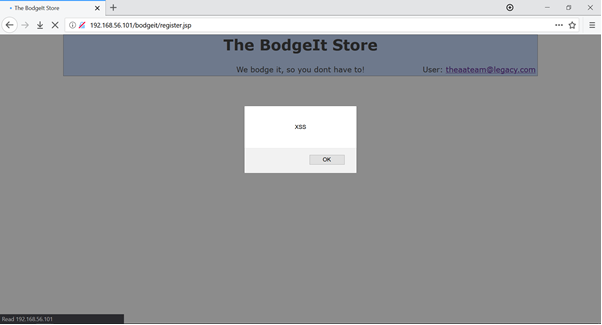


Figure 7: Level 2: Display a popup using: <script>alert("XSS")</script> using sql query via register page

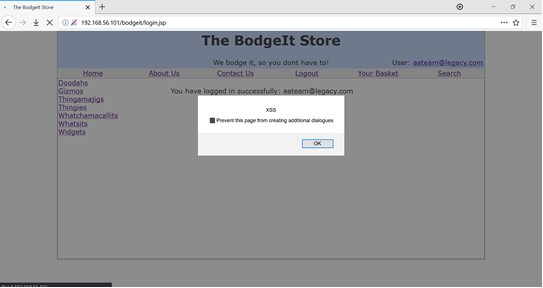


Figure 8 Level 3: Display a popup using: <script>alert("XSS")</script> Via the log in page

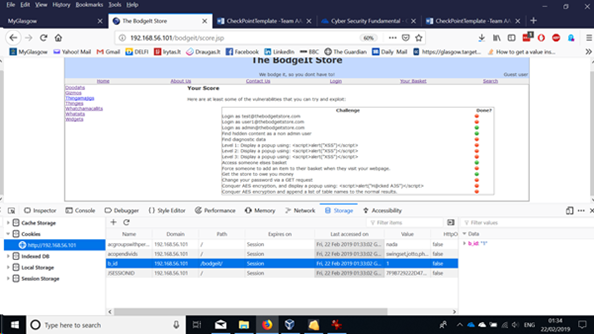


Figure 9 Access someone else's basket by exploiting poor session management control

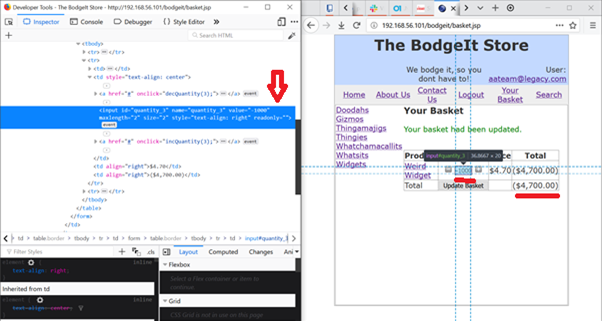


Figure 10 Get the store to owe you money by altering web page code

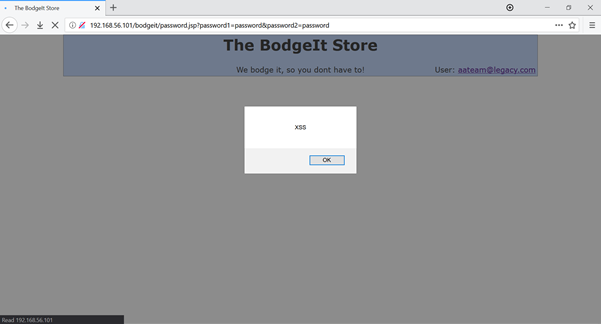


Figure 11 Changing your password via a GET request – new password (“password”) is encoded in URL bar

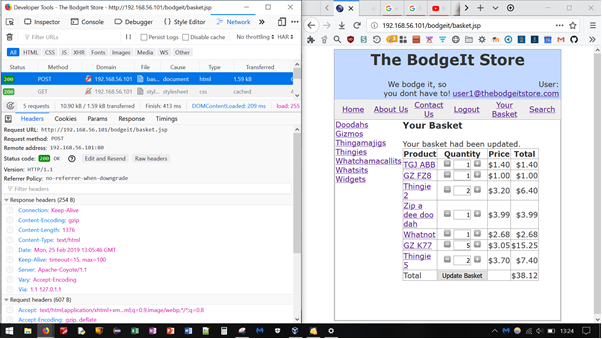


Figure 12 Force someone to add an item to their basket when they visit your webpage.