## Bug Bounty Bootcamp

The Guide to Finding and Reporting Web Vulnerabilities

by Vickie Li

errata updated to print 3

<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>xvi</td>
<td>Mechanisms</td>
<td>Mechanisms</td>
<td>Print 2</td>
</tr>
<tr>
<td></td>
<td><strong>Cooking</strong> Sharing</td>
<td><strong>Cookie</strong> Sharing</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Figure update</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td><code>{ &quot;alg&quot; : &quot;none&quot;, &quot;typ&quot; : &quot;JWT&quot; } { &quot;user&quot; : &quot;admin&quot; }</code></td>
<td><code>{ &quot;alg&quot; : &quot;none&quot;, &quot;typ&quot; : &quot;JWT&quot; } { &quot;user_name&quot; : &quot;admin&quot; }</code></td>
<td>Print 2</td>
</tr>
<tr>
<td>Page</td>
<td>Error</td>
<td>Correction</td>
<td>Print</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| 166  | ```
    def validate_token()
    if (request.csrf_token == session.csrf_token):
        pass
    else:
        throw_error("CSRF token incorrect. Request rejected.")
[...]
    def process_state_changing_action():
        if request.csrf_token:
            validate_token()
        execute_action()
``` | ```
    def validate_token():
        if (request.csrf_token == session.csrf_token):
            pass
        else:
            throw_error("CSRF token incorrect. Request rejected.")
[...]
    def process_state_changing_action():
        if request.csrf_token:
            validate_token()
        execute_action()
``` | Print corrected |

This fragment of Python code first checks whether the CSRF token exists. If it exists, the code will proceed to validate the token. If the token is valid, the code will continue. If the token is invalid, the code will stop the execution and produce an error.

<table>
<thead>
<tr>
<th>203</th>
<th>URL update</th>
<th>You can find it at <a href="https://github.com/digininja/DVWA/">https://github.com/digininja/DVWA/</a></th>
<th>Print 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>For example, a base64-encoded block of XML code tends to start with <code>LD94bWw</code>, which is the base64-encoded string of &quot;&lt;?xml&quot;.</td>
<td>For example, a base64-encoded block of XML code tends to start with <code>PD94bWw</code>, which is the base64-encoded string of &quot;&lt;?xml&quot;.</td>
<td>Pending</td>
</tr>
</tbody>
</table>
| 297  | ```
    Access-Control-Allow-Origin: b.example.com
``` | ```
    Access-Control-Allow-Origin: https://b.example.com
``` | Print 3 |

The application can also return the `Access-Control-Allow-Origin` header with a wildcard character (*) to indicate that the resource on that page can be accessed by any domain.

On the other hand, if the origin of the requesting page isn't allowed to access the resource, the user's browser will block the requesting page from reading the data.

CORS is a great way to implement cross-origin communication. However, CORS is safe only when the list of allowed origins is properly defined. If CORS is misconfigured, attackers can exploit the misconfiguration and access the protected resources.

The most basic misconfiguration of CORS involves allowing the null origin. If the server sets `Access-Control-Allow-Origin` to null, the browser will allow any site with a null origin header to access the resource. This isn't safe because any origin can create a request with a null origin. For instance, cross-site requests generated from a document using the data: URL scheme will have a null origin.

On the other hand, if the origin of the requesting page isn't allowed to access the resource, the user's browser will block the requesting page from reading the data.

CORS is a great way to implement cross-origin communication. However, CORS is safe only when the list of allowed origins is properly defined. If CORS is misconfigured, attackers can exploit the misconfiguration and access the protected resources.

The most basic misconfiguration of CORS involves allowing the null origin. If the server sets `Access-Control-Allow-Origin` to null, the browser will allow any site with a null origin header to access the resource. This isn't safe because any origin can create a request with a null origin. For instance, cross-origin requests generated from a document using the data: URL scheme will have a null origin.
An interesting configuration that isn't exploitable is setting the allowed origins to the wildcard (*). This isn't exploitable because CORS doesn't allow credentials, including cookies, authentication headers, or client-side certificates, to be sent with requests to these pages. Since credentials cannot be sent in requests to these pages, no private information can be accessed:

If not, send a request to the site with the origin header attacker.com, and see if the Access-Control-Allow-Origin in the response is set to attacker.com. (You can add an Origin header by intercepting the request and editing it in a proxy.)

Finally, test whether the site properly validates the origin URL by submitting an Origin header that contains an allowed site, such as www.example.com.attacker.com. See if the Access-Control-Allow-Origin header returns the origin of the attacker's domain.

If not, send a request to the site with the origin header https://attacker.com, and see if the Access-Control-Allow-Origin in the response is set to https://attacker.com. (You can add an Origin header by intercepting the request and editing it in a proxy.)

Finally, test whether the site properly validates the origin URL by submitting an Origin header that contains an allowed site, such as https://www.example.com.attacker.com. See if the Access-Control-Allow-Origin header returns the origin of the attacker's domain.

Figure 19-2: Is the site vulnerable to a CORS misconfiguration vulnerability?
<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>308</td>
<td><strong>Cooking</strong> Sharing</td>
<td><strong>Cookie</strong> Sharing</td>
<td>Print 2</td>
</tr>
</tbody>
</table>