Name: Bharath Karumudi

Lab: Cross Site Request Forgery Attack

Task 1: Observing HTTP Request.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**Observation**: Installed the “HTTP Header Live” addon to the Firefox and opened the Elgg website to examine the HTTP headers.

Explanation: When opened a page, we can see various parameters like host, User-Agent, Referrer, POST and GET requests, cookie details, expiry date and time.

Task 2: CSRF Attack using GET Request

A screenshot of a social media post

Description automatically generated

Fig: Examining the GET request to add a friend.

A screenshot of a cell phone

Description automatically generated

Fig: Confirmation showing “No Friends” in Alice account.

A screenshot of a cell phone

Description automatically generated

Fig: Created a page to attract Alice - vegasSweepStakes

A screenshot of a social media post

Description automatically generated

Fig: Alice opened the vegasSweepstakes page, which is attacker page.

A screenshot of a cell phone

Description automatically generated

Fig: Boby, the attacker was added to Alice friends without her knowledge.

**Observation**: Created a malicious webpage with the GET request and made Alice to click the link. When Alice opened the link, Boby was added to her Friends list.

**Explanation**: This is due to the Cross-Site Request Forgery attack (CSRF). Boby used the CSRF vulnerability and observed how to add a friend and built the GET request which server will accept for Adding a Friend and placed it in a web page. When Alice clicked the link the browser loaded the page, but due to GET request embedded in the malicious page, the browser sent a GET request to the Elgg along with the cookie that is associated with the Alice. The server validated the cookie and considered as a legit request and added the Boby as Alice friend.

Task 3: CSRF Attack using POST Request

A screenshot of a social media post

Description automatically generated

Fig: Examining the POST requests for Profile editing

A screenshot of a social media post

Description automatically generated

Fig: Finding Alice GID

A screenshot of text

Description automatically generated

Fig: Created a new web page to attack Alice and update her profile page

A screenshot of a social media post

Description automatically generated

Fig: Sending a message to Alice with the malicious URL

A screenshot of a social media post

Description automatically generated

Fig: Logged in with Alice and verified no profile descriptions.

A screenshot of a social media post

Description automatically generated

Fig: Received a message from Boby

A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generated

Fig: Alice profile has updated, and the description is publicly visible. Attack successful.

**Observation**: Created a new web page with POST request embedded into it which the server can understand the request to edit the profile. Once Alice clicked the link, her profile description has been updated to “Boby is my Hero” and also it is public.

**Explanation**: This is due to CSRF attack, where Boby examined how Elgg handles the edit profile request. Based on that, he created a malicious web page with the POST request and shared the URL with the Alice. When Alice clicked the URL, the browser sends the POST request to Elgg website and also her cookie. The webserver authorized the change as it is coming from Alice and the profile has been updated.

**Question1**: Boby can just search for members and go to Alice profile. When he hovers the mouse on the “Add Friend” button, the URL will show the ID (friend=) (as shown in second image above). Also, the other way is, by going to the members page and examine the html using developer’s tool, inspector. By this way also, Boby can get GIDs of all users in one place.

**Question2**: It may possible but not straightforward. Because, the malicious site needs to get the GID dynamically onload and it is impossible to get it from Elgg site, as it is cross site. One way is by brute forcing, by running the GIDs in loop, the one that matches will update and for the others, it will say, permission denied. The other way, may be using XSS attack (but not sure at this time, will know in next lab).

Task 4: Implementing a countermeasure for Elgg

A screenshot of a computer

Description automatically generated

Fig: Turned ON the Elgg CSRF countermeasures

A screenshot of a computer

Description automatically generated

Fig: Examining the POST request

A close up of text on a white background

Description automatically generated

Fig: Created a new malicious page with updated description

A screenshot of a social media post

Description automatically generated

Fig: Boby sent a new malicious webpage to update the Alice profile description

A screenshot of a computer

Description automatically generated

Fig: Alice clicked the URL, but no impacted. Attack Failed.

**Observation**: After turning ON the CSRF countermeasures, again observed the HTTP headers and saw the \_\_elgg\_ts and \_\_elgg\_tokens. So updated the malicious page with new headers and performed the attack to update the Alice Profile description as done similar to Task 3, but the attack was failed.

**Explanation:** Elgg now has the timestamp and secret token validations and when Boby tried to attack the Alice with new malicious page even by updating the POST request including the \_\_elgg\_ts and \_\_elgg\_token, it failed. Because the tokens are not matched with the tokens that are with Alice session. Thus, the CSRF attack was defended by the web server and this is a countermeasure for CSRF attacks.