**ISSUE Found**

Absence of anti-csrf token

**CWE ID**

352

**Information: -**

No Anti-CSRF tokens were found in a HTML submission form.

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

CSRF attacks are effective in a number of situations, including:

\* The victim has an active session on the target site.

\* The victim is authenticated via HTTP auth on the target site.

\* The victim is on the same local network as the target site.

CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.

Affected URL

|  |  |
| --- | --- |
| URL | <https://apisetu.gov.in/public/register/orgreg/signup> |
| Method | POST |
| Evidence | <form class="form form-horizontal" ng-submit="submitForm(regForm)" name="regForm" ng-controller="regValidation" method="post" accept-charset="utf-8" novalidate > |
| URL | <https://apisetu.gov.in/public/auth/signin> |
| Method | GET |
| Evidence | <form class="form form-horizontal" ng-submit="submitForm(regForm)" name="regForm" ng-controller="regValidation" method="post" accept-charset="utf-8" novalidate > |
| URL | <https://apisetu.gov.in/signup.php> |
| Method | GET |
| Evidence | <form action='blank.php' class="row" id='form\_subscribe' method="post" name="myForm"> |
| URL | <https://apisetu.gov.in/public/login> |
| Method | GET |
| Evidence | <form action="https://ndh.digitallocker.gov.in/public/login" method="post" accept-charset="utf-8"> |
| URL | <https://apisetu.gov.in/signup.php> |
| Method | GET |
| Evidence | <form name="contactForm" id='contact\_form' class="form-border" method="post" action='blank.php'> |
| URL | <https://apisetu.gov.in/signin.php> |
| Method | GET |
| Evidence | <form name="contactForm" id='contact\_form' class="form-border" method="post" action='blank.php'> |
| URL | <https://apisetu.gov.in/signin.php> |
| Method | GET |
| Evidence | <form action='blank.php' class="row" id='form\_subscribe' method="post" name="myForm"> |
| URL | <https://apisetu.gov.in/public/register/orgreg/signup> |
| Method | GET |
| Evidence | <form class="form form-horizontal" ng-submit="submitForm(regForm)" name="regForm" ng-controller="regValidation" method="post" accept-charset="utf-8" novalidate > |
| URL | <https://apisetu.gov.in/public/auth/signin> |
| Method | POST |
| Evidence | <form class="form form-horizontal" ng-submit="submitForm(regForm)" name="regForm" ng-controller="regValidation" method="post" accept-charset="utf-8" novalidate > |

How to Fix: -

1. Phase: Architecture and Design
2. Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
3. For example, use anti-CSRF packages such as the OWASP CSRFGuard.
4. Phase: Implementation
5. Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.
6. Phase: Architecture and Design
7. Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).
8. Note that this can be bypassed using XSS.
9. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.
10. Note that this can be bypassed using XSS.
11. Use the ESAPI Session Management control.
12. This control includes a component for CSRF.
13. Do not use the GET method for any request that triggers a state change.
14. Phase: Implementation
15. Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

References: -

1. http://projects.webappsec.org/Cross-Site-Request-Forgery
2. http://cwe.mitre.org/data/definitions/352.html