**Audit Report From NIC**

1. **High Risk**
2. **Medium Risk**
3. **Low Risk**
4. **High Risk**

Impact I f the exploitation of the vulnerability can result in complete takeover of the system, alteration/destruction of the data , disclosure of sensitive information. Exploitability Exploitation for the vulnerability is easy and can be done using tools/scripts that are easily available or are open-source.

**SQL Injection**

Do not dynamically construct and execute query strings instead use prepared statements or parameterized queries. Wherever possible, use a strong white-list for validating user input be fore passing them to SQ L queries. Grant the database user minimum privileges that are required to accomplish the tasks.

**Malicious File Upload**

Application should check allowed File extension and File type (MIME Type) in the upload modul using white-list filter at server side . File to be uploaded should be restricted to a particular size. Server-side check for not allowing long filename with double ex tension/double -dot(.)/nullbyte (%00)/meta characters. Assign only Read and W rite permissions to the upload folders as required."

**Privilege Escalation**

"Create an access control matrix and map each role of the application as per the transactions allowed for each role . The authentication and authorization policies be role based, to minimize the effort required to maintain these policies. The policies should be highly configurable, in order to minimize any hard coded aspects of the policy. The enforcement mechanism(s) should deny all access by de fault, requiring explicit grants to specific users and roles for access to every page. I f the page is involved in a work flow, check to make sure the conditions are in the proper state to allow access."

**Cross Site Scripting (XSS)**

Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters , and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-de fined regular expression. Input which fails the validation should be rejected, not sanitized. U ser input should be HTML-encoded at any point where it is copied into application responses. All HTML m eta characters, including < > "" ' and =, should be replaced with the corresponding HTML entities (< > etc). In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax

**Cross Site Request Forgery**

Use a CSRF guard code . A C SR F guard code is a server side code that inserts a hidden random value in the requested page of a web application. W hen that page is resubmitted to the web server with some user input, this hidden value is verified by the C SRF guard code. I f the resubmitted page contains the hidden value, it is allowed through for processing. If the hidden value is not present, the C SRF guard blocks that page with the user input. Use POST instead of GET requests . Even though the attack shown here was carried out on a PO ST request; forging fake POST requests is much harder than forging GET requests. Another countermeasure that should be considered is using the referrer header field to validate the origin of the request. Even though it can be faked it makes it m ore difficult for the attacker.

**Brute Force Attack**

Implement rate limiting on login request. Implement a strong alphanumeric C APTCHA that is at-least 6 characters long. Lock out a targeted account

**Insecure Direct Object Reference**

"Avoid exposing your private object references to users whenever possible, such as primary keys or filenames. Instead of sending primary keys (like code in the above scenario) in URL, better to use session to send such a kind of information. This should be implemented for all the parameters used across all the pages in the application. Validate any private object references extensively with an ""accept known good"" approach. Verify authorization to all referenced objects. I f you must expose direct references to database structures, ensure that SQ L statements and other database access methods only allow authorized records to be shown."

**Clear Text Submission of Password**

"Use latest stable version of TLS to protect all sensitive communications passing between the client and the server. Implement HTTP Strict Transport Layer Security. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP. If implementing HTTPS is not possible, implement salted SHA256 or salted SHA-512 hashing algorithm s on password fields, while using plain SHA-256 or SHA-512 hashing on new password fields."

**Medium Risk**

**CSV Injection**

"To remediate it, ensure that no cells begin with any of the following characters: • Equals to (“=”) • Plus (“+”) • Minus (“-“) • At (“@”) • Tab (0x09) • Carriage return (0x0D) Alternatively, prepend each cell field with a single quote, so that their content will be read as text by the spreadsheet editor”.

**Improper Error Handling**

Ensure that a customized error message is shown for any error that has occurred, which gives out very limited information. Disable or limit detailed error handling. In particular, do not display debug information to end users, stack traces, or path information. Application should make secure to prevent revealing of any kind of error and Hardening process should be carried out periodically.

***No Lockout Policy***

The following solutions are recommended for the above mentioned flaw:

1. The user should be blocked after a failure of three login attempts. The blocked user can be unblocked by a user with administrative privileges or can be unblocked after certain period of time.

2. Alternatively implement C APTCHA at login page

**Session Fixation**

Invalidate the existing session and assign a new session identifier on successful login attempts. For platform s such as ASP that do not generate new values for session id cookies, utilize a secondary cookie. In this approach, set a secondary cookie on the user's browser to a random value and set a session variable to the same value. I f the session variable and the cookie value ever don't match, invalidate the session, and force the user to log on again.

***Weak Password Policy***

Enforce strong password requirement Minimum Length: 8 characters At-least 1 uppercase, 1 lowercase , 1 numeric and 1 special characters . Restrictions against password reuse Restrictions against using common passwords Restrictions against using contextual string in the password (e .g., user id, app name)

**Insufficient Logging & Monitoring**

Information to be logged includes the following: IP address of the originating Source , Date , Time, Username (No Password), , Referrer, Process id, UR L, User Agent, Countries if any in addition to other details to be logged in the website . Logging of Authentication Process which includes number of successful and failed login attempts. To create audit logs , use auto numbering so that every logged entry has an un-editable log number. Then if one audit entry is deleted a gap in the numbering sequence will appear. Report of the website logs to be generated weekly by the administrator to keep track of the website activities.

**Directory Listing**

Directory browsing should be disabled on web server.

Using Components with Known Vulnerabilities

Upgrade the components used to the latest version. If upgradation is not possible , download the security patches and hide the version details

**Low Risk**

**Lack of Cookie Attributes**

Upgrade the components used to the latest version. If up gradation is not possible , download the security patches and hide the version details.

**Weak/Improper Authentication**

Validation of Username Case-sensitivity must be check before accepting the values.

**Server Banner Disclosure**

Server Version should not be display in response of application.

Note :- Server side problem (Rajat)

**Click Jacking**

Preventing the browser from loading the page inframe using the X-Frame-Options or Content Security Policy (frameancestors) HTTP headers. Preventing session cookies from being included when the page is loaded in a frame using the Sam e Site cookie attribute . Implementing JavaScript code in the page to attempt to prevent it being loaded in a frame (known as a ""frame-buster"").

**Captcha**

Not Implemented Properly Implement captcha and ensures that captcha is implemented properly.

**Security Headers Not Implemented**

Implement security headers such as X-XSS-Protection, C ontent-Security-Policy, R e ferrer Policy, X-Content-TypeOptions, Permission Policy and Strict -transport-layerprotection