Phase 3: Testing and Usability

Team:
Soha Bhatia (sbhatia6)
Akruti Sinha (asinha6)
Shivangi Chopra (schopra4)

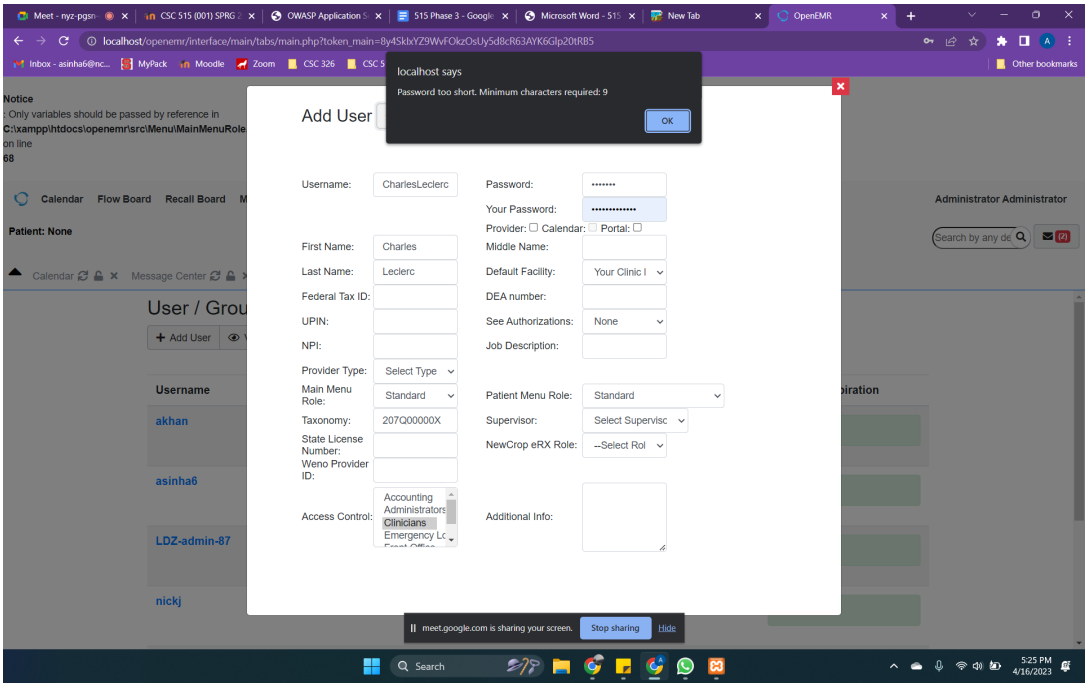
Part 1:

V 2.1.1 Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).

Checking the System -

To verify that user-set passwords in OpenEMR are at least 12 characters in length (after multiple spaces are combined), you can follow these steps:

1) Access the OpenEMR system as an administrator or privileged user.
2) Navigate to the user management section of the system where you can view and manage user accounts.
3) Identify the password field for each user account and view the existing password or allow the user to set a new password.
4) Check the length of the password to ensure that it is at least 12 characters long, after combining multiple spaces.
5) If the password is less than 12 characters in length, it prompts the user to change their password to meet the minimum password length requirement.



As we can see from the prompt, the user set passwords are not at least 12 characters in length (after multiple spaces are combined) but instead should be just 9 characters in length.

```
var pwdresult = passwordvalidate(document.forms[0].clearPass.value);
if(pwdresult == 0) {
        flag=1;
        alert(<?php echo xlj('The password must be at least eight characters, and should'); ?> -
        '\n' +
        <?php echo xlj('contain at least three of the four following items:'); ?> +
        '\n' +
        <?php echo xlj('A number'); ?> +
        '\n' +
        <?php echo xlj('A lowercase letter'); ?> +
        '\n' +
        <?php echo xlj('An uppercase letter'); ?> +
        '\n' +
        <?php echo xlj('A special character'); ?> +
        '\n' +
        '(' +
        <?php echo xlj('not a letter or number'); ?> +
        ').' +
        '\n' +
        <?php echo xlj('For example:'); ?> +
        ' healthCare@09');
        return false;
```

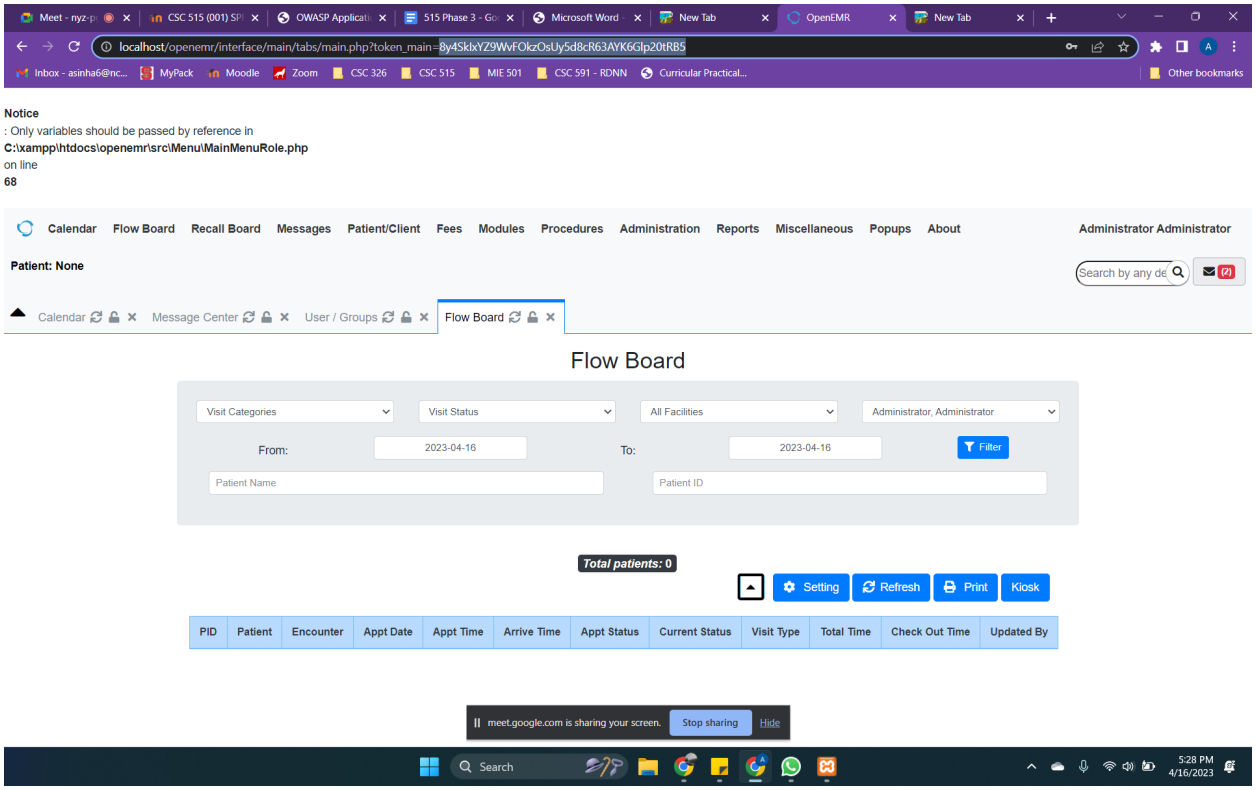
<u>Conclusion</u> : Security requirements not met.

V 3.1.1 Verify the application never reveals session tokens in URL parameters.

Checking the System -

1. Access the OpenEMR system as an administrator or privileged user.
2. Navigate to the login page and log in to the system.
3. Once logged in, we check the URL of the current page to ensure that it does not include any session tokens or other sensitive information.
4. Navigate to other pages within the system and repeat step 3 to ensure that session tokens are not revealed in the URL parameters.

However, we could not find any instances where session tokens were being revealed in URL parameters This might be because OpenEMR must have implemented secure session management practices, such as using secure cookies to store session information and ensuring that session tokens are never included in URL parameters.



Conclusion:

As we see from the above experiments and UI screenshot, that the security requirement V 3.1.1 is being met

V 3.3.1 Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.

Checking the System -

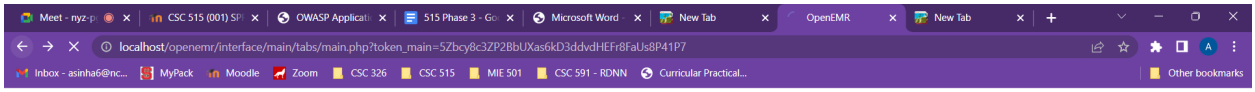
We may run the following tests to ensure that the session token is invalidated on logout or expiration:

1. Logout functionality is tested by first logging in to the application and then clicking the logout button to check that the session is invalidated and the user is logged out. Then, we attempt to return to the application by using the browser's back button or reloading the page. The user should be sent to the login page or an error page if the session is correctly invalidated.
2. Experiment with session expiration by logging into the program and leaving it inactive for a period of time until the session expires. Following that, we attempt to perform an authentication-required action, such as visiting a restricted page or submitting a form. If the user's session has expired as expected, he or she should be returned to the login page or an error page.

These tests will help us confirm that, upon logout or expiration, the session token is appropriately destroyed, preventing the back button or a downstream relying party from restarting an authenticated session, including across relying parties.

Conclusion:

We found that OpenEMR does not resume an authenticated session on clicking back button, however, we could see a warning message flash briefly (1-2 seconds) before redirecting to the login page.



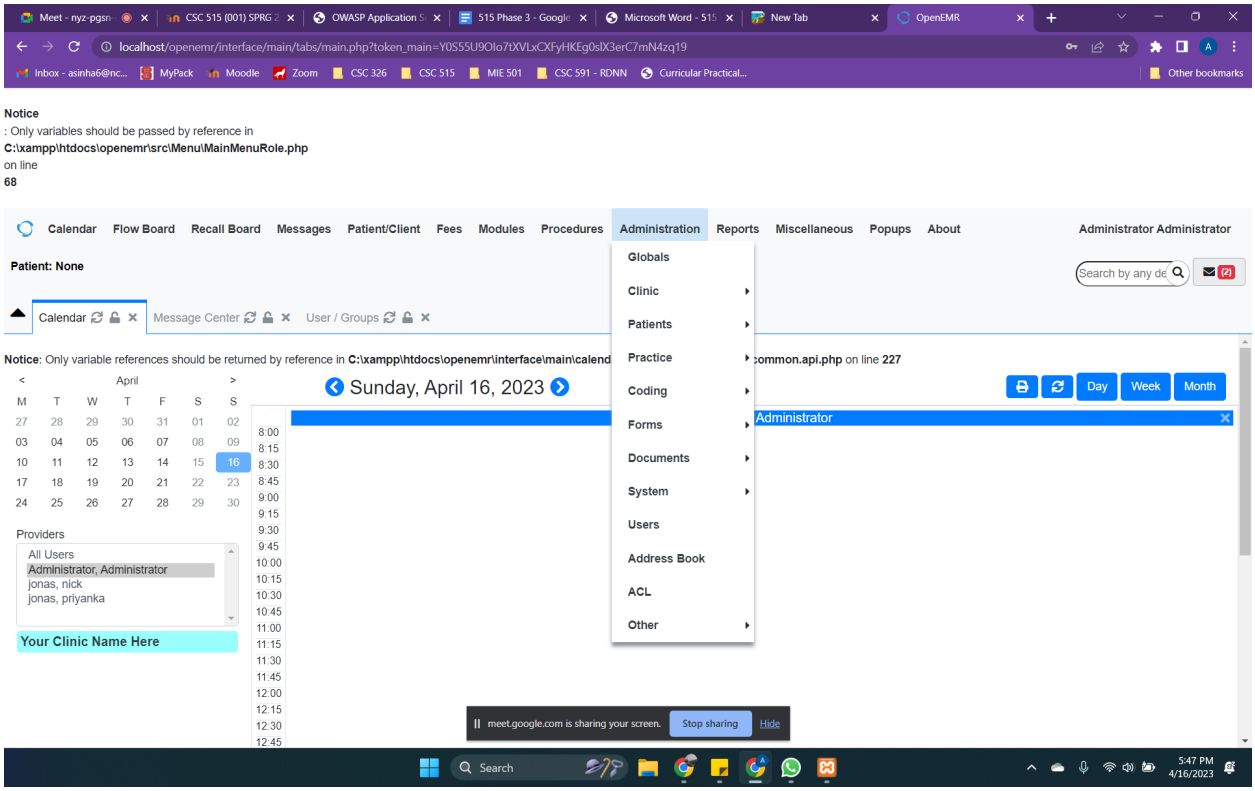
V4.1.1 Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.

Checking the system -

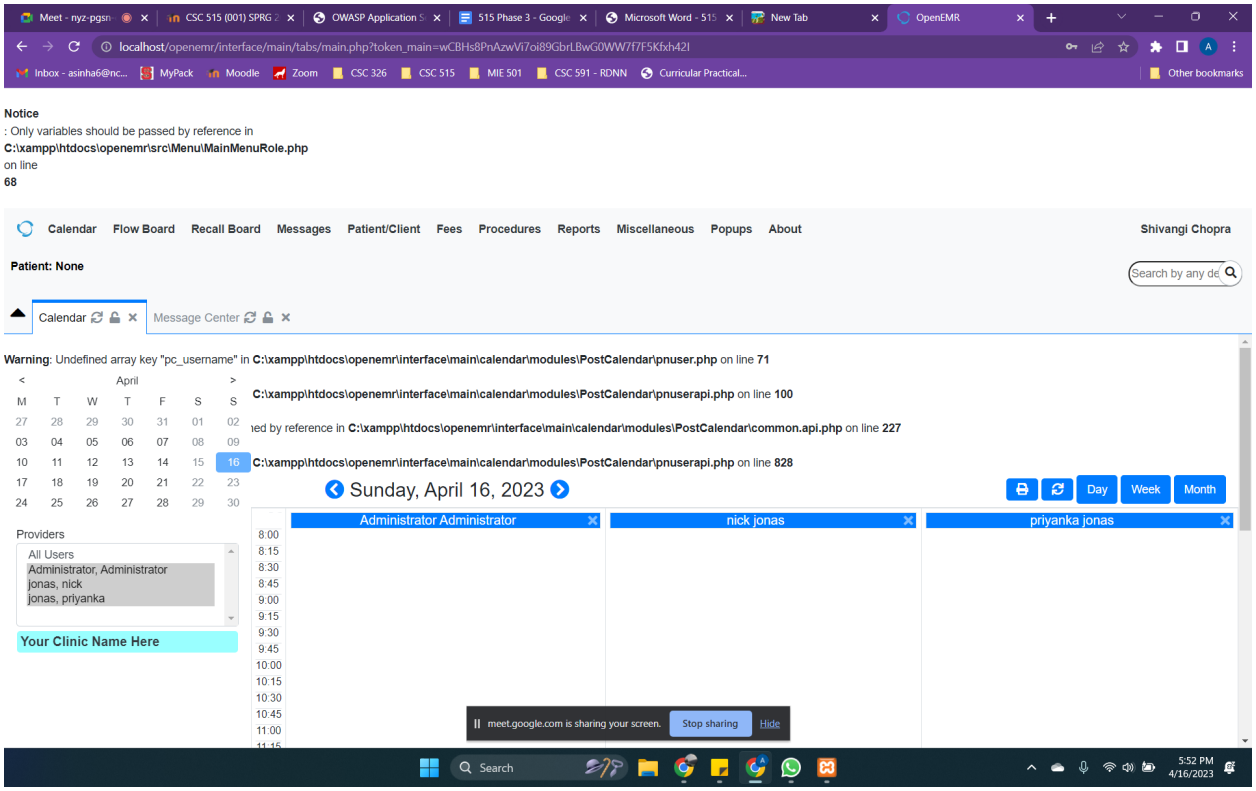
ADMIN:

1) We first checked the access control implementation on the client side.
2) When we login as an admin , we see that all the options on the pge are enabled.
3) Then when we login as any other non-privileged user; we see that on the client side a lot of options were disabled in the HTML for certain modules.
4) We tried to manipulate the HTML and made those modules enabled by just changing the styling. However, still those options were not clickable and no request was being sent to the server.

Login as Admin:



Login as Physician:



5) Now considering that the client side checks are being bypassed , it is important to find out if proper server-side checks are in place.
6) This calls for checking the codebase and finding out if proper authorization is managed in php.
7) The "interface/main/auth/auth.inc.php" file provides procedures for managing user roles, login, and logout as well as user authentication and authorisation.

   https://github.com/openemr/openemr/blob/master/interface/main/authorizations/authorizations.php

8) Global variables and functions pertaining to user sessions and access control, such as the "$GLOBALS['authorized']" variable and the "acl_check" function, are contained in the "interface/globals.php" file.
9) Functions for user authentication and authorization, such as password hashing, token creation, and user role management, are contained in the "library/auth.inc.php" file.
10) Access control lists (ACLs), which are used to define user roles and permissions for various system components, contain functions and definitions in the "library/acl.inc" file.
11) User groups are used to organize users with similar responsibilities and permissions, and the "interface/usergroup/usergroup_admin.php" file provides methods that let administrators administer user groups.

<u>Conclusion</u> : Not confident about weather security requirements met or not as -

1) We were able to perform a CSRF attack before by manipulating the cookie of the admin and changing the modAction in the POST request.
2) This might have been caused due to liniency in handling the session cookies. Although proper access control is in check for requests sent to the server. However, if the request itself was manipulated the session cookie management is not properly checked. This might be due to lack of proper encryption and hashing mechanisms to protect the session cookie or session IDs that are not invalidated on logout or after a period of inactivity.
3) To make sure that only cookies belonging to Admin or privileged users are permitted to enable/disable registered modules, the openEMR cookie must be compared to the "modAction" argument given in the POST request

V5.1.3 Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists)

Checking the system -

To verify and check if all the input is is validated using positive validation, we performed the following steps:

1. Reviewed the application code and documentation to identify all input sources including HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, and any other sources of input that we checked in the application .
2. We then verified that each input source is validated using positive validation techniques, such as allow lists. Allow lists specify what input values are allowed and can help prevent attacks like SQL injection, cross-site scripting (XSS), and command injection.
3. We then Tested the application using various input values to ensure that the allow lists are working properly. For example, we tried entering special characters, invalid input values, and known attack payloads to see if they are properly rejected by the application.
4. For URL Params we Understood the expected format of the URL parameter. For example, if the parameter is expected to be an integer,we tried to verify that only integers are accepted and processed.
5. Further we Verified that the REST API rejects requests that are not on the allowed list, such as requests to disallowed endpoints or requests using disallowed request methods

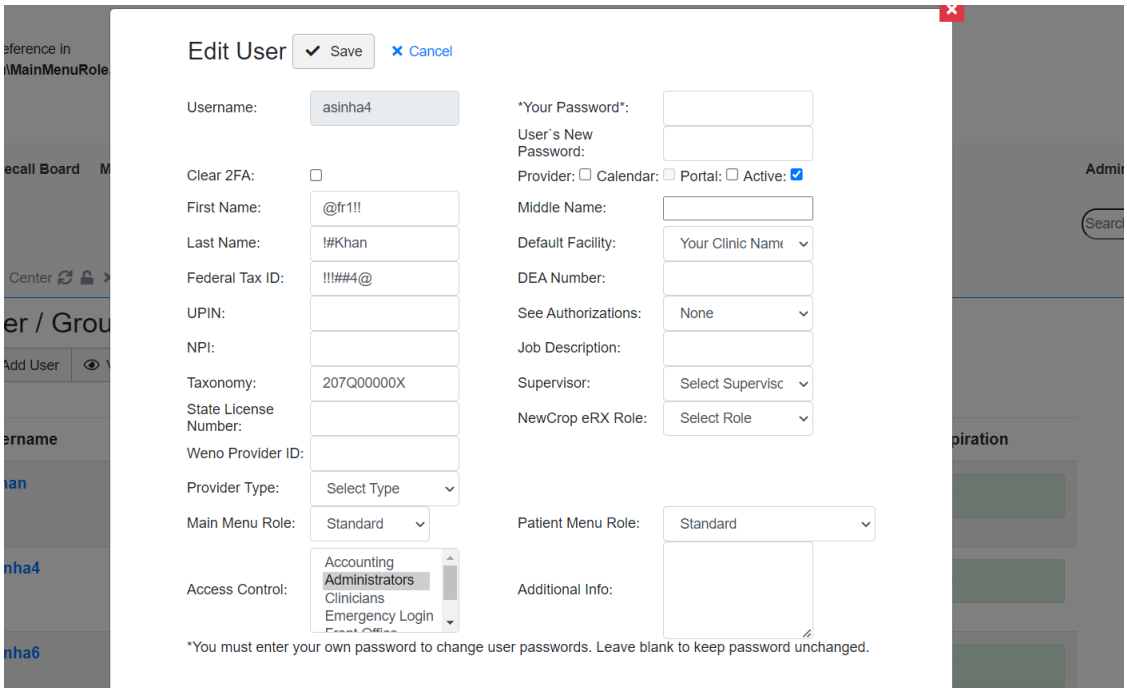
<u>Conclusion</u>:

Following is the application code where the REST API URL is being verified for unwanted values.

```
// for both REST API and browser access we can't proceed unless we have a valid site id.
// since this is user provided content we need to escape the value but we use htmlspecialchars instead
// of text() as our helper functions are loaded in later on in this file.
if (empty($tmp) || preg_match('/[^A-Za-z0-9\\-.]/', $tmp)) {
    echo "Invalid URL";
    error_log("Request with site id '" . htmlspecialchars($tmp, ENT_QUOTES) . "' contains invalid characters.");
    die();
}
```

However, we could see that some HTML Input form fields were able to take unwanted characters in some fields and no proper checks were in place for input validation.

Below we see that the name field and Federal TAX ID field is accepting unallowed characters .



Username	Real Name	Additional Info	Authorized	MFA	Password Expiration
akhan	Aryan Khan		no	no	2023-10-07
asinha4	@fr1!! !#Khan		no	no	2023-10-19
asinha6	Akruti Sinha		no	no	2023-10-07

```
for(i=0;i<f.length;i++){
  if(f[i].type=='text' && f[i].value)
  {
    if(f[i].name == 'fname' || f[i].name == 'mname' || f[i].name == 'lname')
    {
      alertMsg += checkLength(f[i].name,f[i].value,35);
      alertMsg += checkUsername(f[i].name,f[i].value);
    }
    else if(f[i].name == 'taxid')
    {
      alertMsg += checkLength(f[i].name,f[i].value,10);
      alertMsg += checkFederalEin(f[i].name,f[i].value);
    }
    else if(f[i].name == 'state_license_number')
    {
      alertMsg += checkLength(f[i].name,f[i].value,10);
      alertMsg += checkStateLicenseNumber(f[i].name,f[i].value);
    }
    else if(f[i].name == 'npi')
    {
      alertMsg += checkLength(f[i].name,f[i].value,10);
      alertMsg += checkTaxNpiDea(f[i].name,f[i].value);
    }
    else if(f[i].name == 'drugid')
    {
      alertMsg += checkLength(f[i].name,f[i].value,30);
      alertMsg += checkAlphaNumeric(f[i].name,f[i].value);
    }
```

The code for these fields also didn't have many checks corresponding to allow lists.
So V5.1.3 requirement is not met in the system.

V6.1.2 Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records

Checking the system :

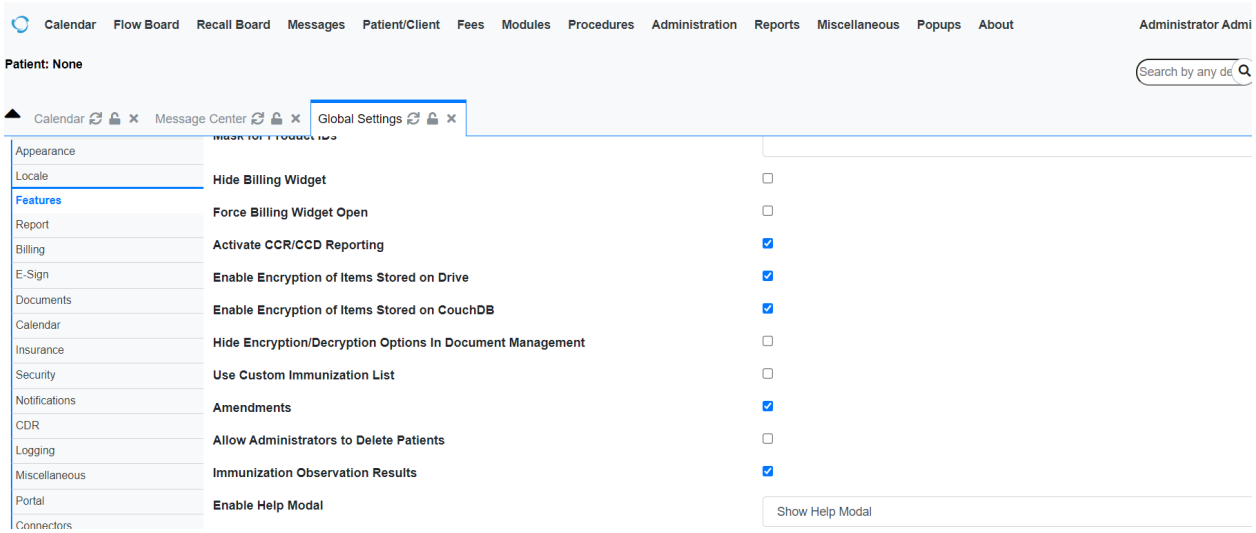
To perform the verification of the requirement, we performed the following steps:

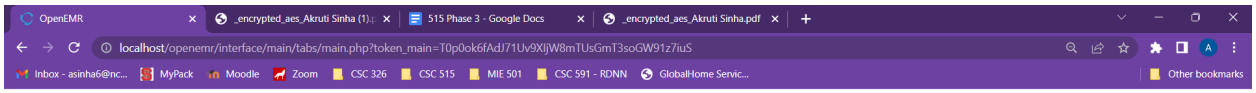
1. Review the storage policies to find out how regulated health data is being stored and what encryption techniques are in use, examine the application's data storage rules.
2. Then, to make sure that the data storage systems are configured securely and that regulated health data is being appropriately protected, we audited them. Tools that manually or automatically examine data storage systems for vulnerabilities can be used to accomplish this, we did this by accessing the MySQL database where logged in to the MySQL server, and selected the OpenEMR database
3. We also verified the various security options available to the admin and users to encrypt the health records being stored in the system.

Conclusion:

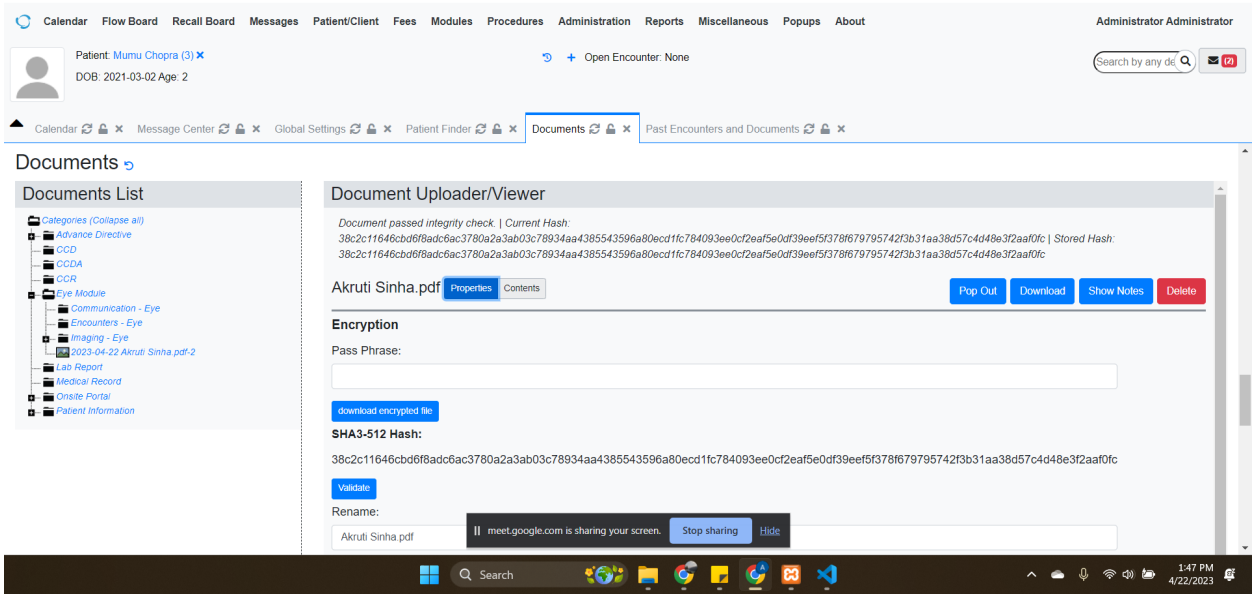
1. The document encryption feature currently available in OpenEMR , simply allows the user to download an encrypted form of the document and also allows the user to upload an encrypted document, which is then stored in OpenEMR in an unencrypted form
2. However according to our findings there is no support to encrypt all patient data in the database and the patient documents. So, perhaps an OS based encryption method should be used .

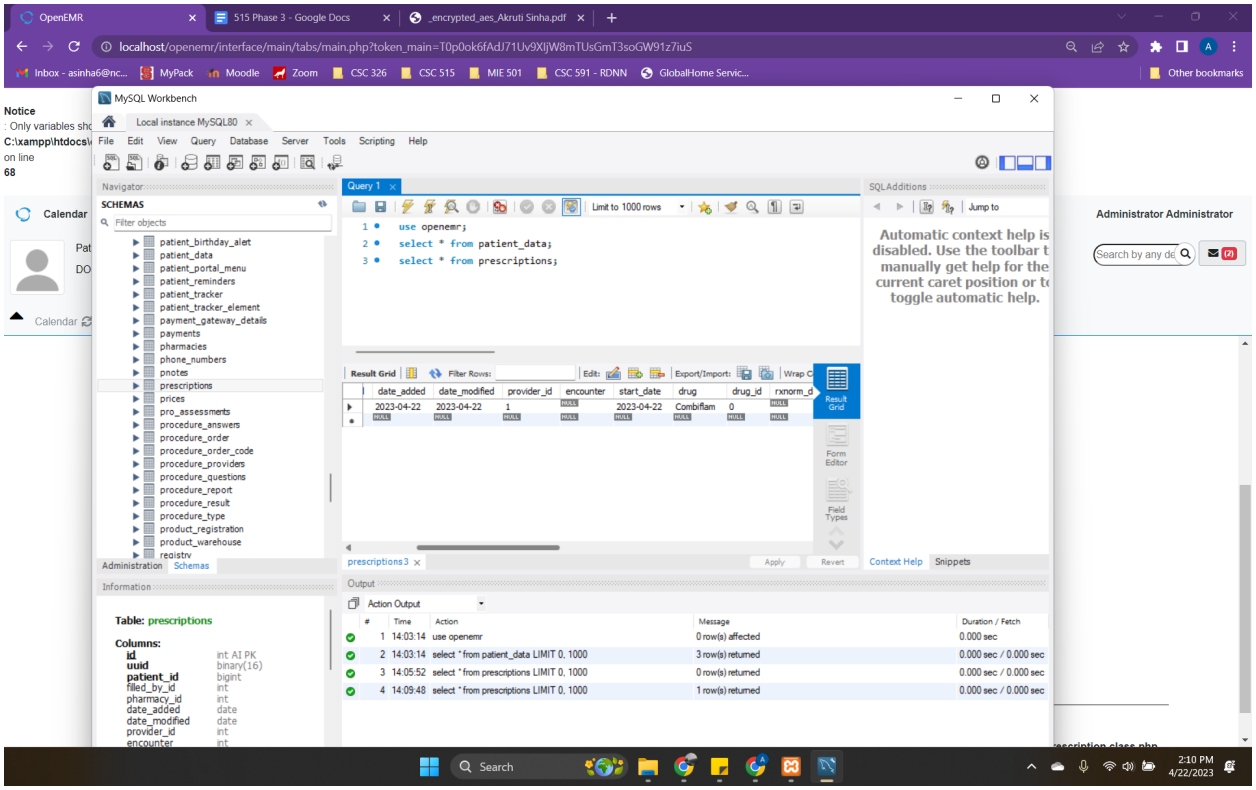
lotice
Only variables should be passed by reference in
:\xampp\htdocs\openemr\src\Menu\MainMenuRole.php
n line
8

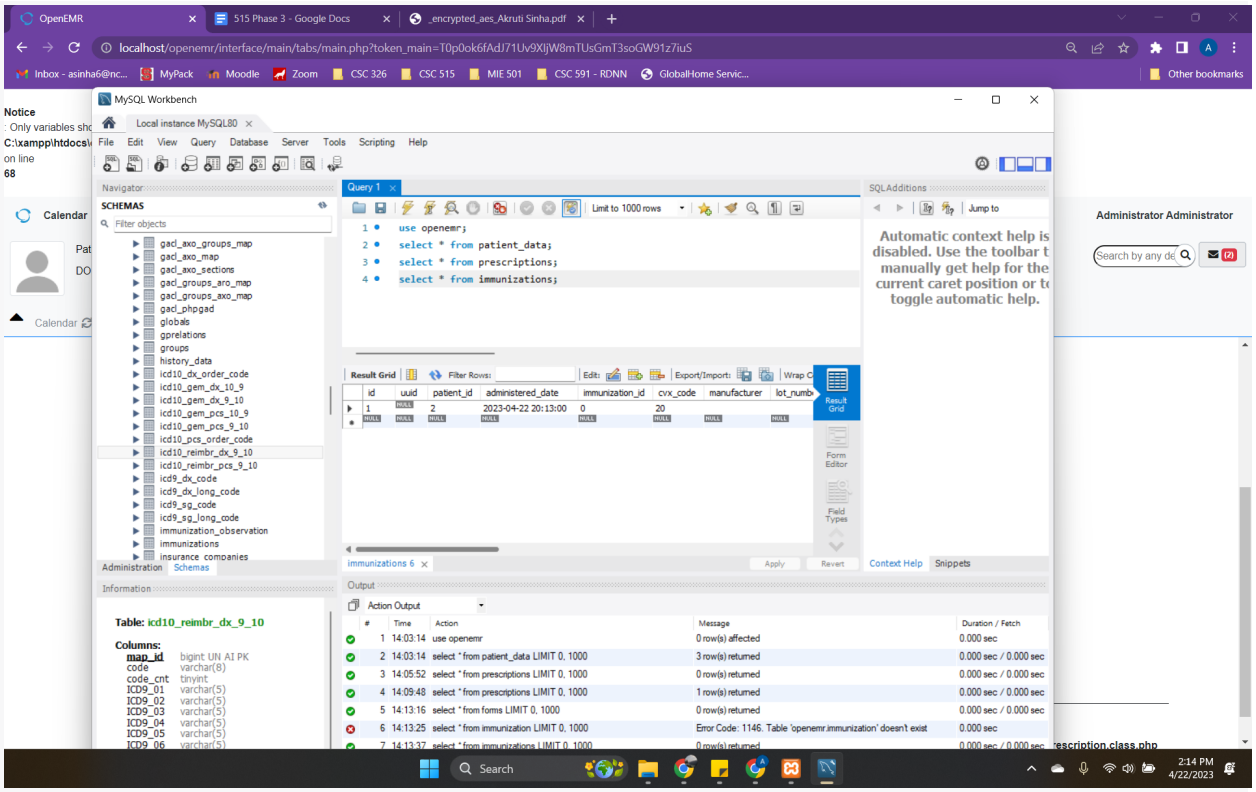




Notice
: Only variables should be passed by reference in
C:\xampp\htdocs\openemr\src\Menu\MainMenuRole.php
on line
68




OpenEMR
515 Phase 3 - Google Docs
_encrypted_aes_Akruti Sinha.pdf
localhost/openemr/interface/main/tabs/main.php?token_main=T0p0ok6fAdJ71Uv9XljW8mTUsGmT3soGW91z7iuS
MySQL Workbench
Local instance MySQL80
use openemr;
select * from patient_data;
select * from prescriptions;
Table: prescriptions
Columns:
id int AI PK
uuid binary(16)
patient_id bigint
filled_by_id int
pharmacy_id int
date_added date
date_modified date
provider_id int
encounter int
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Administrator Administrator
2:10 PM
4/22/2023


OpenEMR
515 Phase 3 - Google Docs
_encrypted_aes_Akruti Sinha.pdf
localhost/openemr/interface/main/tabs/main.php?token_main=T0p0ok6fAdJ71Uv9XljW8mTUsGmT3soGW91z7iuS
MySQL Workbench
Local instance MySQL80
use openemr;
select * from patient_data;
select * from prescriptions;
select * from immunizations;
Table: icd10_reimbr_dx_9_10
Columns:
map_id bigint UN AI PK
code varchar(8)
code_cnt tinyint
ICD9_01 varchar(5)
ICD9_02 varchar(5)
ICD9_03 varchar(5)
ICD9_04 varchar(5)
ICD9_05 varchar(5)
ICD9_06 varchar(5)
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Administrator Administrator
2:14 PM
4/22/2023

V6.2.2 Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography

The application uses cryptographic methods to encrypt various secrets and passwords throughout the application.

Checking the system -

1) We walked through the code base of OpenEMR to discover the various libraries and methods used to encrypt the secrets.

https://github.com/openemr/openemr/blob/master/src/Common/Crypto/CryptoGen.php

2) One such custom component used by OpenEMR is CryptoGen().

3) This class offers strategies for creating random keys and initialization vectors (IVs), with various cryptographic techniques.

4) To produce keys and IVs for symmetric encryption methods like AES as well as asymmetric encryption techniques like RSA, the class leverages the openssl extension in PHP. Additionally, the class offers ways to produce random strings and bytes for use in cryptography.

5) The CryptoGen class supports both symmetric encryption methods like AES, which are widely used in the business, and asymmetric encryption techniques like RSA. These algorithms are extensively used and have undergone extensive research and analysis by subject matter experts.

```
145      */
146     private function coreEncrypt($sValue, $customPassword = null, $keySource = 'drive', $keyNumber = null)
147     {
148         $keyNumber = isset($keyNumber) ? $keyNumber : $this->keyVersion;
149
150         if (!extension_loaded('openssl')) {
151             error_log("OpenEMR Error : Encryption is not working because missing openssl extension.");
152         }
153
154         if (empty($customPassword)) {
155             // Collect the encryption keys. If they do not exist, then create them
156             // The first key is for encryption. Then second key is for the HMAC hash
157             $sSecretKey = $this->collectCryptoKey($keyNumber, "a", $keySource);
158             $sSecretKeyHmac = $this->collectCryptoKey($keyNumber, "b", $keySource);
159         } else {
160             // customPassword mode, so turn the password into keys
161             $sSalt = RandomGenUtils::produceRandomBytes(32);
162             if (empty($sSalt)) {
163                 error_log('OpenEMR Error : Random Bytes error - exiting');
164                 die();
165             }
166             $sPreKey = hash_pbkdf2('sha384', $customPassword, $sSalt, 100000, 32, true);
167             $sSecretKey = hash_hkdf('sha384', $sPreKey, 32, 'aes-256-encryption', $sSalt);
168             $sSecretKeyHmac = hash_hkdf('sha384', $sPreKey, 32, 'sha-384-authentication', $sSalt);
169         }
170
171         if (empty($sSecretKey) || empty($sSecretKeyHmac)) {
172             error_log("OpenEMR Error : Encryption is not working because key(s) is blank.");
173         }
174
175         $iv = RandomGenUtils::produceRandomBytes(openssl_cipher_iv_length('aes-256-cbc'));
176         if (empty($iv)) {
177             error_log('OpenEMR Error : Random Bytes error - exiting');
178             die();
```

```
180
181         $processedValue = openssl_encrypt(
182             $sValue,
183             'aes-256-cbc',
184             $sSecretKey,
185             OPENSSL_RAW_DATA,
186             $iv
187         );
188
```

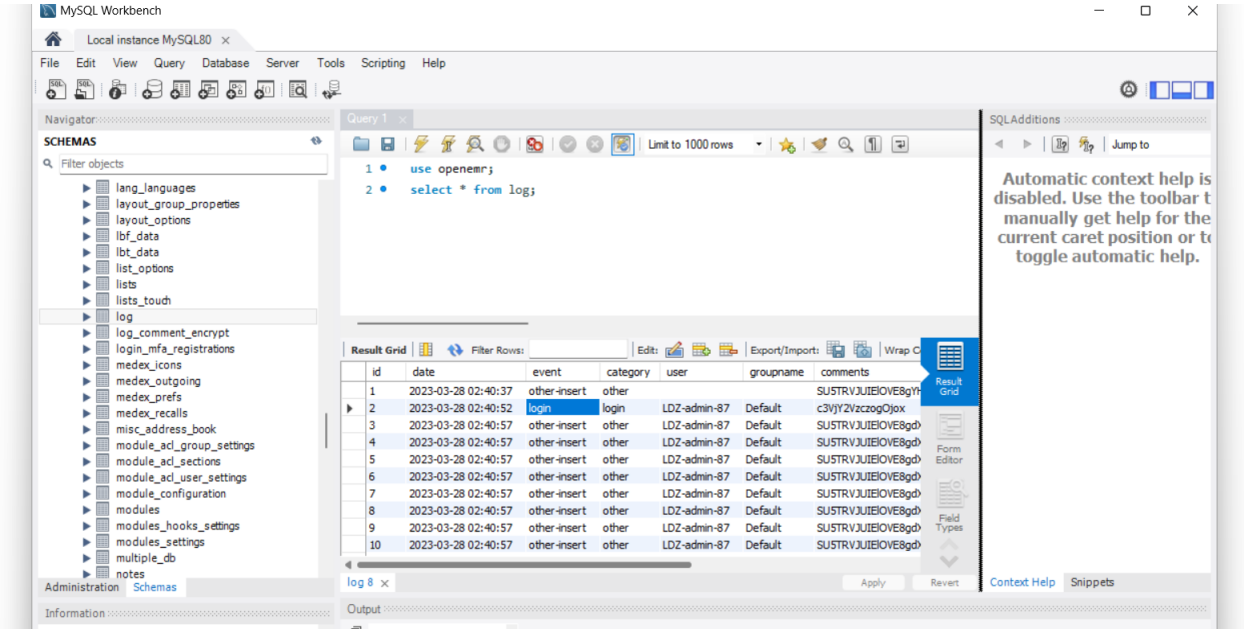
Conclusion: Security Requirement met.

V7.1.3 Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures.

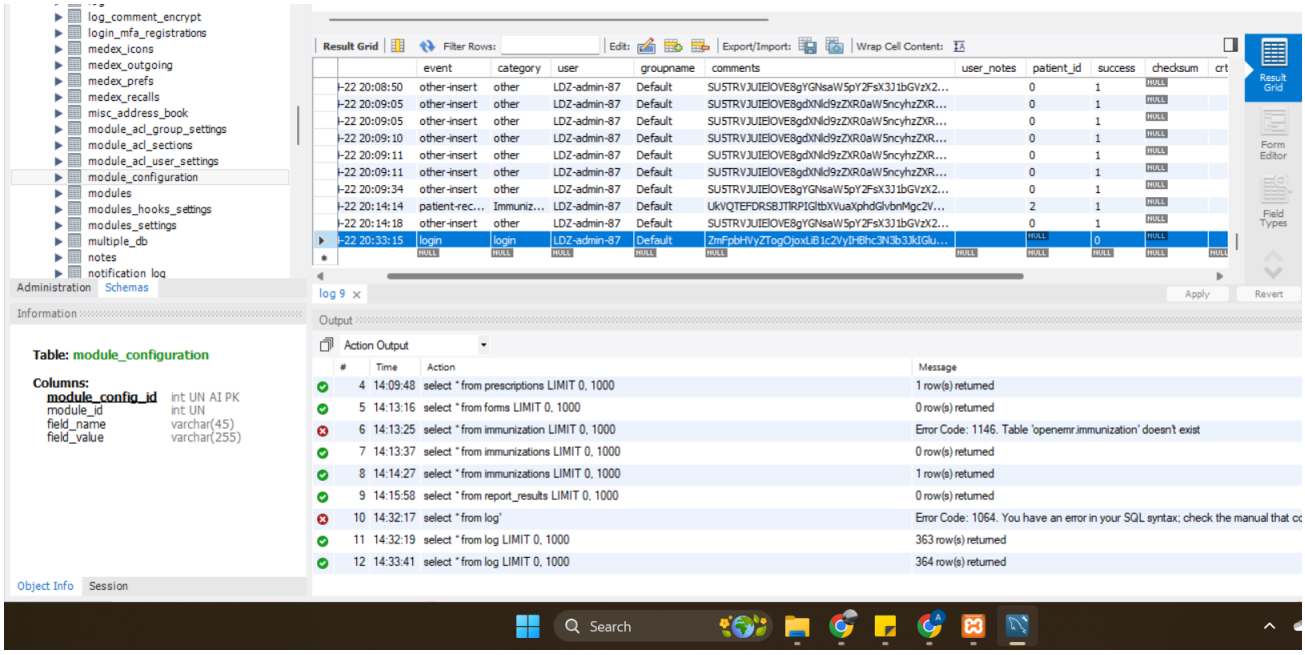
Checking the System :-

To verify that the application maintains logs of all security relevant events ,we performed the following steps :

1. Log in to the application with a user account.
2. Attempt to access a resource that requires higher privileges than the current user account has.
3. Attempt to input invalid data into a form field or parameter.
4. We then checked the application's logs to ensure that security-relevant events, such as access control failures, input validation failures, and deserialization failures, have been logged.
5. We also verified that successful and failed authentication events have also been logged.



Invalid authentication is also logged:



Conclusion: V7.1.3 requirement is being met in our system

V7.1.4 Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.

Checking The System

When an event occurs in OpenEMR, we can perform the following procedures to ensure that each log event contains the essential information for a full study of the timeline:

1. Define the required information: First, we must establish what information is required for a thorough study of the timeline when an event occurs. This may include the event's timestamp, the individual or system responsible for the event, the action taken, any relevant patient data, and the event's outcome.
2. Examine the OpenEMR logging options: Next, we should double-check OpenEMR's logging settings to ensure that all relevant information is being collected in the logs. OpenEMR logs events to the Apache error log by default, but we may configure it to log events to a different log file if that is more convenient for examination.
3. Check for data accuracy: We should guarantee that the data in the logs is valid and trustworthy by ensuring that the system clock is synchronized with a reliable time source and that the user and system identities are correct. It is also critical to validate the accuracy of any data or parameters collected.
4. Test the logs: We can put the logs through their paces by simulating an inquiry using the data from the logs. This can aid in the identification of any missing information or gaps in the log data. To search the logs for specific occurrences or patterns, we can use tools like grep or awk.

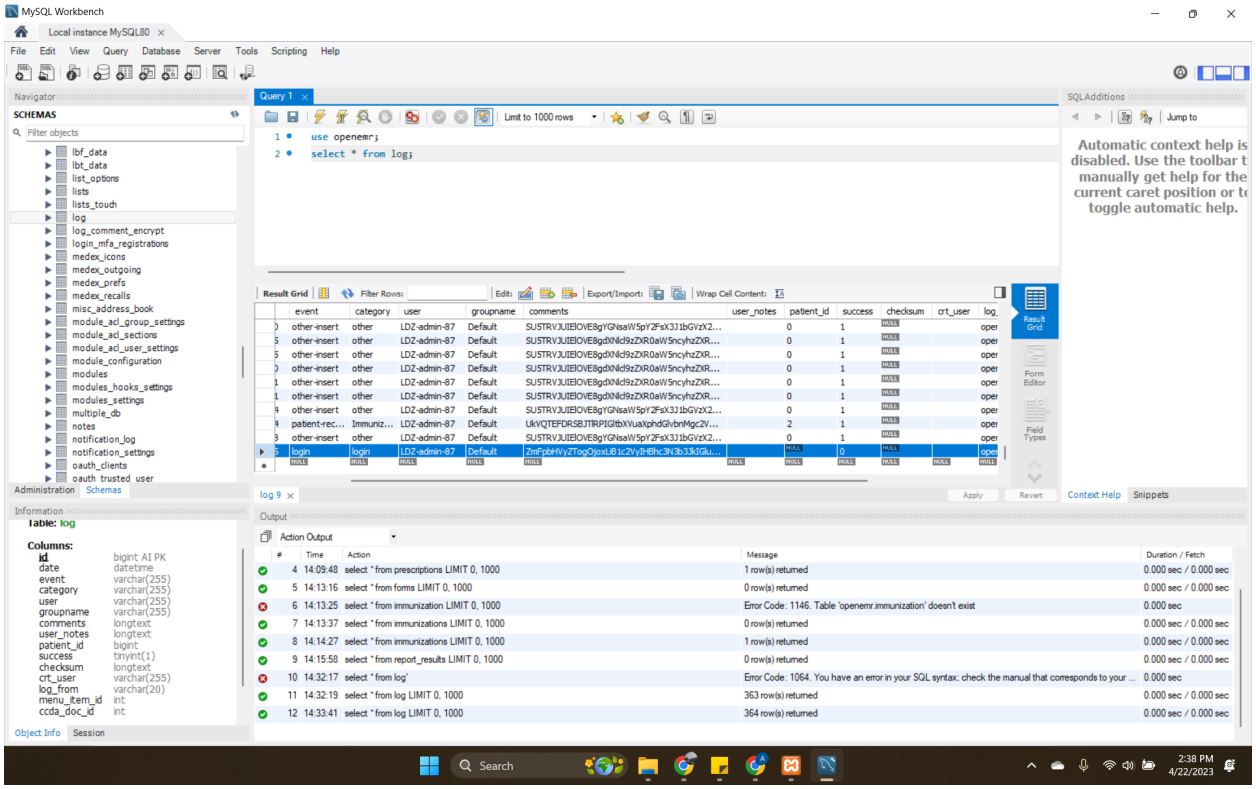
5. Monitor and review: We should put in place a mechanism for monitoring and reviewing the OpenEMR logs on a regular basis to ensure that they continue to capture the relevant information and remain accurate and dependable over time. To centralize and analyze logs, log management technologies such as Graylog or ELK can be employed.

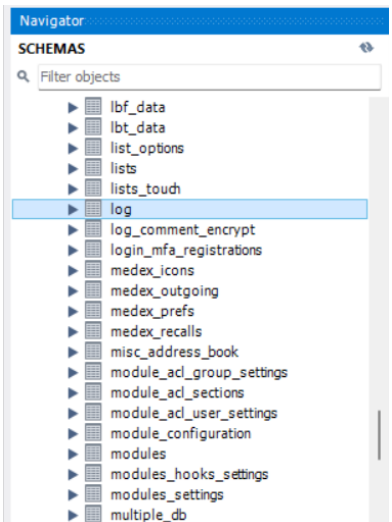
By following these procedures, we can ensure that each log event contains the required information for a full study of the chronology when an event occurs in OpenEMR, allowing us to examine any issues that develop swiftly and effectively.

The following fields can be found in the log table in the database: id, date, event, category, user, groupname, comments, user_notes, patient_id, success, checksum, crt_user, log_from, menu_item_id, ccda_doc_id.

Conclusion:

The logging is insufficient since it does not adhere to the 5Ws and 1H standard practices for logging. Furthermore, the comments are irrelevant.

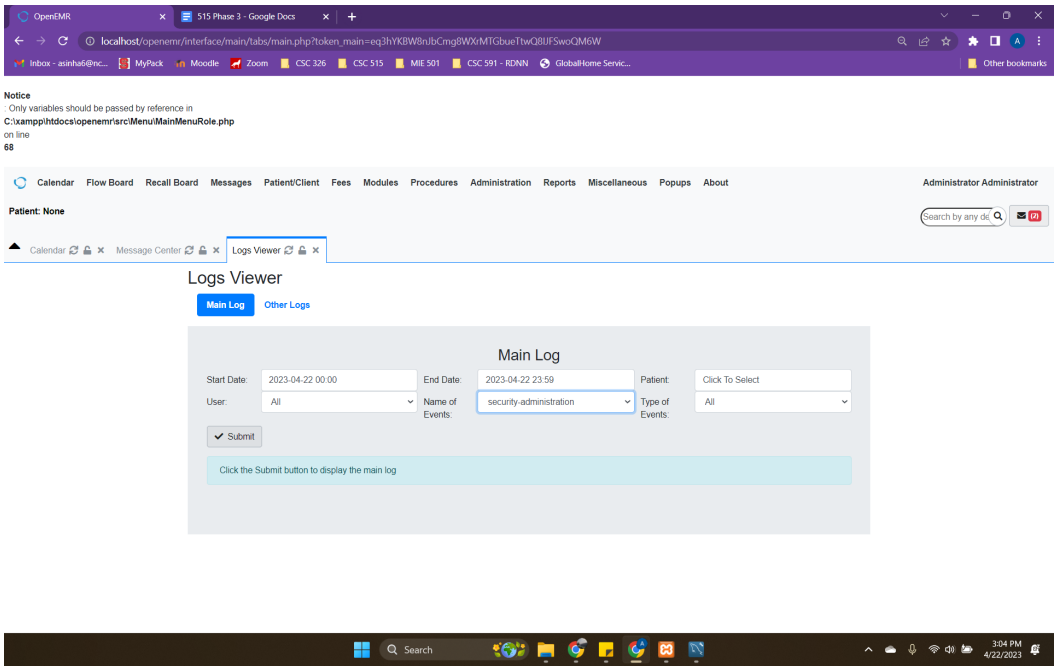




V7.3.3 Verify that security logs are protected from unauthorized access and modification.

Checking the system -

1) Log in to OpenEMR with an administrative account.
2) Select "Administration" from the top menu bar's tabs.
3) Select "Logs" from the drop-down menu by clicking.
4) Select the security-administration in “Name of Events”.
5) Submit to view the contents.
6) Now if we log in as any other user security log is not accessible to unauthorized users who do not have the necessary permissions. Check that the security logs are listed among the available logs.



<u>Conclusion</u> : Security requirements met.

Part 2:

Jakob Nielson's Ten Usability Heuristics

1. **Visibility of system status**: The ability of a system to offer clear and timely input to users about what is happening is referred to as system visibility. OpenEMR, a web-based electronic medical records (EMR) system, does an excellent job in this regard by presenting users with visible indicators indicating system condition. OpenEMR does well in providing feedback to the user on what is happening, such as displaying a loading bar when a page is loading. When a page in OpenEMR is loading, for example, a loading bar appears to alert the user that the system is processing their request. This feedback reduces the user's perceived wait time and provides assurance that their request is being executed.
2. **Match between system and real world**: The match between system and real world is an important feature of usability that relates to how well a system represents the user's real-world language, concepts, and tasks. This means that, in the case of OpenEMR, the system should be intended to be straightforward and familiar to medical professionals who use it to handle patient data and other clinical responsibilities. OpenEMR does well in using language that is familiar to medical professionals, and the layout of the system resembles a typical patient chart. The system, for example, employs terms like "patient chart," "chief complaint," and "vital signs" that are recognizable to healthcare practitioners. This reduces cognitive strain and confusion, allowing users to concentrate on the work at hand without the need to acquire a new language or vocabulary.
3. **User control and freedom**: An essential part of usability is user control and freedom, which relates to how well a system allows users to govern their interactions with the system and recover from faults or mistakes. This means that in the case of OpenEMR, users should be able to undo actions, travel freely between pages, and recover from mistakes without losing any data or progress. OpenEMR allows users to undo actions and return to previous pages easily. For example, if a user makes a mistake or changes their mind about doing a specific action, they can easily undo it by utilizing the "Undo" feature. This allows users to easily repair faults without having to restart the procedure from the beginning.
4. **Consistency and standards**: Consistency and standards can make it easier for consumers to use OpenEMR since they will know what to anticipate from various interface elements. Users are helped when related functions are referred to using the same language, such as when "Add Patient" or "Edit Patient," respectively. Users can more easily recognize various sorts of information, such as alerts or notifications, by using consistent colors or icons. Additionally, following accepted design guidelines and best practices will enhance OpenEMR's usability and accessibility for a larger spectrum of users. OpenEMR may be made more user-friendly and effective by, for instance, utilizing clear and simple language, providing feedback to user activities, and minimizing needless complexity.
5. **Error prevention**: OpenEMR does provide validation checks on user input to prevent errors, such as requiring a valid date format.
6. **Reduce Memory Load-Recognition rather than recall**: OpenEMR displays patient information and relevant data in a visible and easy-to-access manner, reducing the need for users to recall information.
7. **Tailor- Flexibility and efficiency of use**: Tailorability, flexibility, and efficiency of usage refer to how well a system allows users to adjust the system to their individual demands and work style, and how this customisation improves their system efficiency. In the context of OpenEMR, this

means that users should be able to customize their dashboard and frequently-used actions to improve their efficiency when working with patient records. OpenEMR allows users to customize their dashboard and frequently-used actions, improving efficiency.

8. **Just The Facts-Aesthetic and minimalist design**: Users may find it challenging to locate the information or features they want due to an overly complicated or crowded user interface. In order to produce a user-friendly interface, aesthetic and minimalist design uses legible text, white space, and a limited number of visuals or pictures.OpenEMR does not adhere to this rule particularly UI elements for instance, lowering the amount of buttons or links on a page or streamlining the navigation menu.
9. **Recoverability-Help users recognize, diagnose, and recover from errors**: OpenEMR provides clear error messages and suggestions for recovery, but there is some room for improvement in terms of providing more detailed explanations.
10. **Help and documentation**: OpenEMR provides extensive documentation and support, including user manuals and forums.

NEAT/SPRUCE mnemonic analysis

Warning message: "Access Denied: You do not have sufficient privileges to perform this action."

Warning :Access Denied
NEAT analysis:

- N: Necessary - The warning message is necessary to inform the user of the access denial.
- E: Explained - The warning message explains the reason for the access denial.
- A: Actionable - The warning message suggests a possible course of action, which is to contact the system administrator.
- T: Tested - The warning message has been tested and proven to work as expected.
- Explanation: "Patient demographics are not available for this patient."

Explanation :You do not have sufficient privileges to perform this action
SPRUCE analysis:

- S: Succinct - The explanation is concise and to the point.
- P: Pertinent - The explanation is relevant to the user's action of trying to access patient demographics.
- R: Relevant - The explanation is meaningful and understandable to the user.
- U: User-friendly - The explanation is written in clear and simple language.
- C: Contextual - The explanation is provided in the appropriate context, on the page where the user is trying to access patient demographics.
- E: Exemplary - The explanation sets an example for good user interface design.

Part 3:

Test Case ID: TC-5.2.5-01
ASVS Control :V 5.2.5 : **Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.**

Description: This test case verifies that the application protects against template injection attacks by ensuring that any user input being included in a template is sanitized or sandboxed, and does not allow malicious code injection. It ensures that input validation and sanitization measures are implemented to prevent template injection attacks.

Instructions:

- Login to the system with a valid user account.
- Go to the create patient page.
- Enter the following string in the first name, last name field: ' OR 1=1--
- Submit the form.

Expected Results: The system should reject the input and display an error message.
CWE-94: Improper Control of Generation of Code ('Code Injection')

Test Case ID: TC-2.1.6-02
ASVS Control: V2.1.6 -Verify that password change functionality requires the user's current and new password.

Description: This test case verifies that the application's password change functionality requires the user's current and new password, and does not allow password changes without proper authentication. It ensures that users cannot change passwords without proper authorization, and that the application properly enforces password policies when changing passwords.

Instructions:

- Log in to the application using a valid username and password.
- Navigate to the "Change Password" page or section of the application.
- Attempt to change the password without providing the current password.
- Verify that the application returns an error message stating that the current password is required.
- Provide the current password and attempt to change the password without providing a new password.
- Verify that the application returns an error message stating that a new password is required.
- Provide a new password that does not meet the password policy requirements, such as a password that is too short or lacks complexity.
- Verify that the application returns an error message stating that the new password does not meet the password policy requirements.
- Provide a new password that meets the password policy requirements and verify that the password is changed successfully.
- Log out and attempt to log back in with the old password.
- Verify that the application does not allow access with the old password and requires the new password.

Expected Results:
When running this test case, the application should properly enforce password change functionality by requiring the user's current and new password, verifying that the new password meets the password policy requirements, and preventing unauthorized access with the old password. If any of the steps fail, the test case should be considered a fail.
CWE-620: Unverified Password Change

Test Case ID: TC-3.2.1-03
ASVS Control :V 3.2.1-Verify the application generates a new session token on user authentication.

Description: This test case verifies that the application generates a new session token on user authentication. It ensures that the application generates a new session token for each user and that the token is not shared between different sessions. This helps prevent session hijacking and other session-related attacks.

Instructions:

- Log in to the application with a valid username and password.
- Open the browser's developer tools and navigate to the application's cookies.
- Identify the session token cookie, typically named "token_main" in URL query parameter..
- Note the value of the session token cookie when logging in for the first time.
- Log out of the application and close the browser.
- Reopen the browser and navigate to the application's login page.
- Log in to the application again with the same username and password as before.
- Open the browser's developer tools and navigate to the application's cookies.
- Verify that the session token cookie value has changed from the previous value.
- Repeat the test with different user accounts and verify that each user has a unique session token.

Expected Results:
When running this test case, the application should generate a new session token on user authentication. Each user should have a unique session token that is not shared between different sessions. The session token should change on each login to prevent session hijacking and other session-related attacks. If the session token does not change or is shared between different sessions, the test case should be considered a failure.

CWE-384: Session Fixation

Test Case ID: TC-5.2.8-04
ASVS Control :V5.2.8 Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.

Description: This test case verifies that OpenEMR sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content. It ensures that the application does not allow untrusted user-supplied content to execute code or perform other malicious activities.

Instructions:

- Login as administrator to the OpenEMR application through the login page.
- After login, go to the patients tab and click to create a new patient.
- The website will navigate to the Medical Record Dashboard screen after adding a new patient.
- To enter the patient's medical information, click the "edit" button next to the Allergies option. Next select the Add option to enter the patient's medical conditions connected to allergies.
- Place the provided payload in this title box after selecting the problem type and clicking the save button.
- Payload -
- <svg onload="javascript:alert('ATTACK,ATTACK,ATTACK!!')" xmlns="#"></svg>
- The attached payload will then be processed under the Reports page of the Dashboard, which presents the patient's information as a report.
- And it will be displayed as an alert to the user.

Expected Result:

When running this test case, OpenEMR should sanitize, disable, or sandbox user-supplied scriptable or expression template language content. The application should not allow untrusted user-supplied content to execute code or perform other malicious activities. If the injected content does not execute or cause any other issues, the test case should be considered a pass.

CWE-94: Improper Control of Generation of Code ('Code Injection')

Test Case ID: TC-7.4.1-05

ASVS Control :V7.4.1 Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.

Description: Verify that a generic message is shown when an unexpected or security-sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.

Instructions:

- Log in to the OpenEMR application with valid credentials.
- Navigate to the Miscellaneous
- Navigate to the Blank Forms and select Referral
- Verify that a generic error message is displayed with a unique ID that support personnel can use to investigate.

Expected Results:

A generic error message should be displayed with a unique ID that support personnel can use to investigate. The error message should not reveal any sensitive information about the application, server, or system.

CWE-210: Self-generated Error Message Containing Sensitive Information

Test Case ID: TC-2.1.8-06

ASVS Control :V.2.1.8 Verify that a password strength meter is provided to help users set a stronger password.

Description:

This test case aims to verify that OpenEMR provides a password strength meter to help users set a stronger password. A password strength meter is a visual indicator that displays the strength of the user's password. It evaluates the strength of the password and provides feedback to the user, indicating whether the password is weak, moderate, or strong.

Instructions:

- Launch OpenEMR in a web browser.
- Navigate to the password change or reset screen.
- Enter a password in the password field.
- Observe the password strength meter.
- Verify that the password strength meter is displayed and functional.
- Enter passwords of varying strengths and verify that the strength meter provides accurate feedback.
- Attempt to submit a password that is considered weak by the strength meter and verify that Open EMR prompts the user to choose a stronger password.
- Attempt to submit a password that is considered strong by the strength meter and verify that Open EMR accepts the password.
- Test Data: A weak password such as "password123"

Expected Results:

- The password strength meter is displayed and functional.
- The strength meter provides accurate feedback for passwords of varying strengths.
- Open EMR prompts the user to choose a stronger password if the password is weak.
- Open EMR accepts the password if it is considered strong.

CWE-521: Weak Password Requirements

Test Case ID: TC-3.2.2-07

ASVS Control : V.3.2.2 - Verify that session tokens possess at least 64 bits of entropy

Test Description:

This test case verifies that session tokens generated by OpenEMR have at least 64 bits of entropy. Session tokens are used to maintain user sessions on web applications and require a high level of entropy to ensure they cannot be guessed or brute-forced by an attacker.

Instruction:

- Log in to OpenEMR.
- Navigate to a page that requires authentication.
- Open the browser developer tools and select the "Network" tab.
- Verify that the session token is included in the HTTP request headers.

- Copy the session token and paste it into a text editor.
- Use an online entropy calculator to determine the number of bits of entropy in the session token.
- Verify that the session token has at least 64 bits of entropy.
- Test Data: A session token obtained from OpenEMR.

Expected Results:
The session token is included in the HTTP request headers and the session token has at least 64 bits of entropy.

CWE-331: Insufficient Entropy

Test Case ID: TC-14.4.6-08
ASVS Control : V.14.4.6 Verify that a suitable Referrer-Policy header is included to avoid exposing sensitive information in the URL through the Referer header to untrusted parties.

Description:
This test case aims to verify that OpenEMR includes a suitable Referrer-Policy header to prevent sensitive information in the URL from being exposed through the Referer header to untrusted parties. The Referrer-Policy header can control how much referrer information is included in the Referer header when navigating between pages, thus preventing sensitive information leakage.

Instructions:

- Launch OpenEMR in a web browser.
- Navigate to any page on OpenEMR.
- Open the browser developer tools and select the "Network" tab.
- Click on a link or button that takes you to another page within OpenEMR.
- Observe the Referer header in the request headers for the new page.
- Verify that the Referrer-Policy header is present in the response headers.
- Verify that the Referrer-Policy header is set to an appropriate value such as "strict-origin-when-cross-origin" or "same-origin".

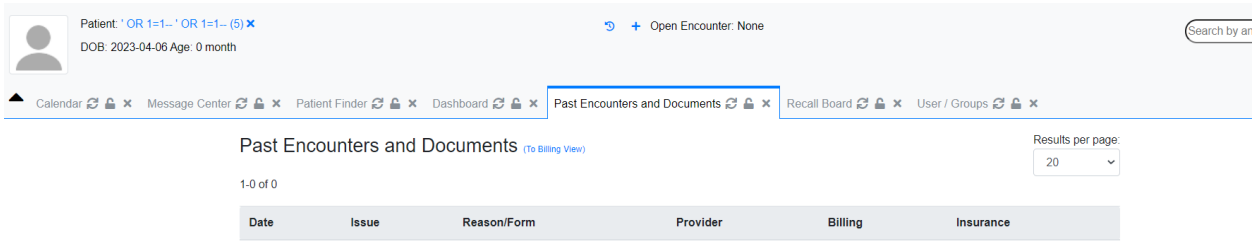
Expected Results:
The Referrer-Policy header is set to an appropriate value such as "strict-origin-when-cross-origin" or "same-origin".

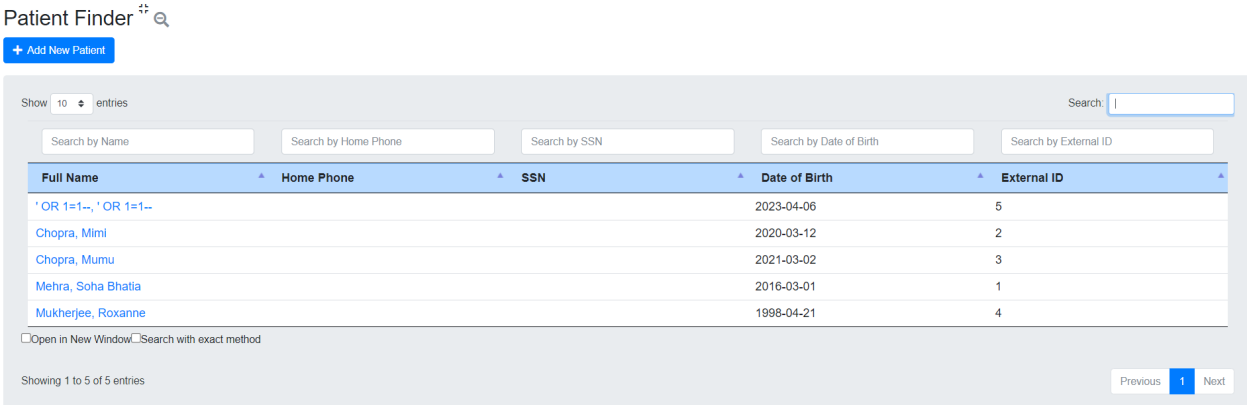
CWE-116: Improper Encoding or Escaping of Output

ACTUAL RESULTS

Test Case ID: TC-5.2.5-01

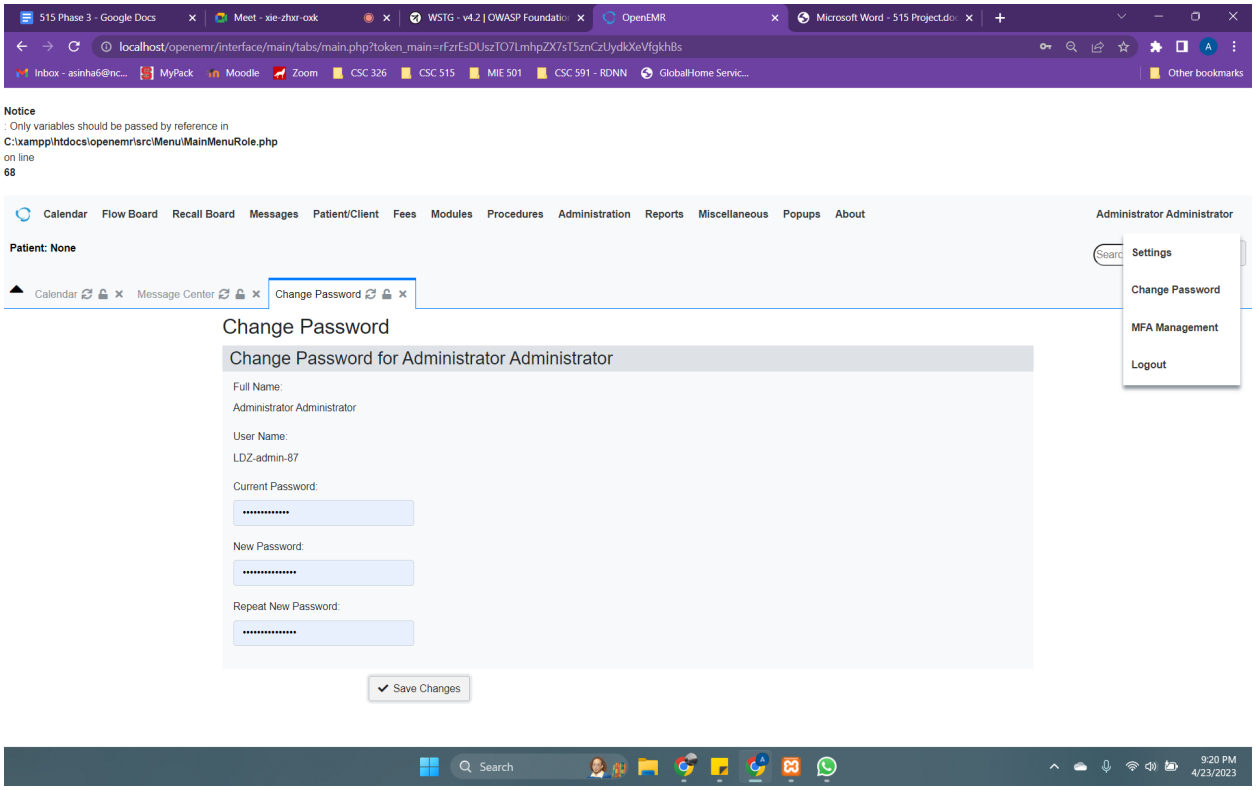
Actual Results: The system did not reject the input or display an error message. The patient was created. CWE: CWE-20: Improper Input Validation. The test case fails.





Test Case ID: TC-2.1.6-02:

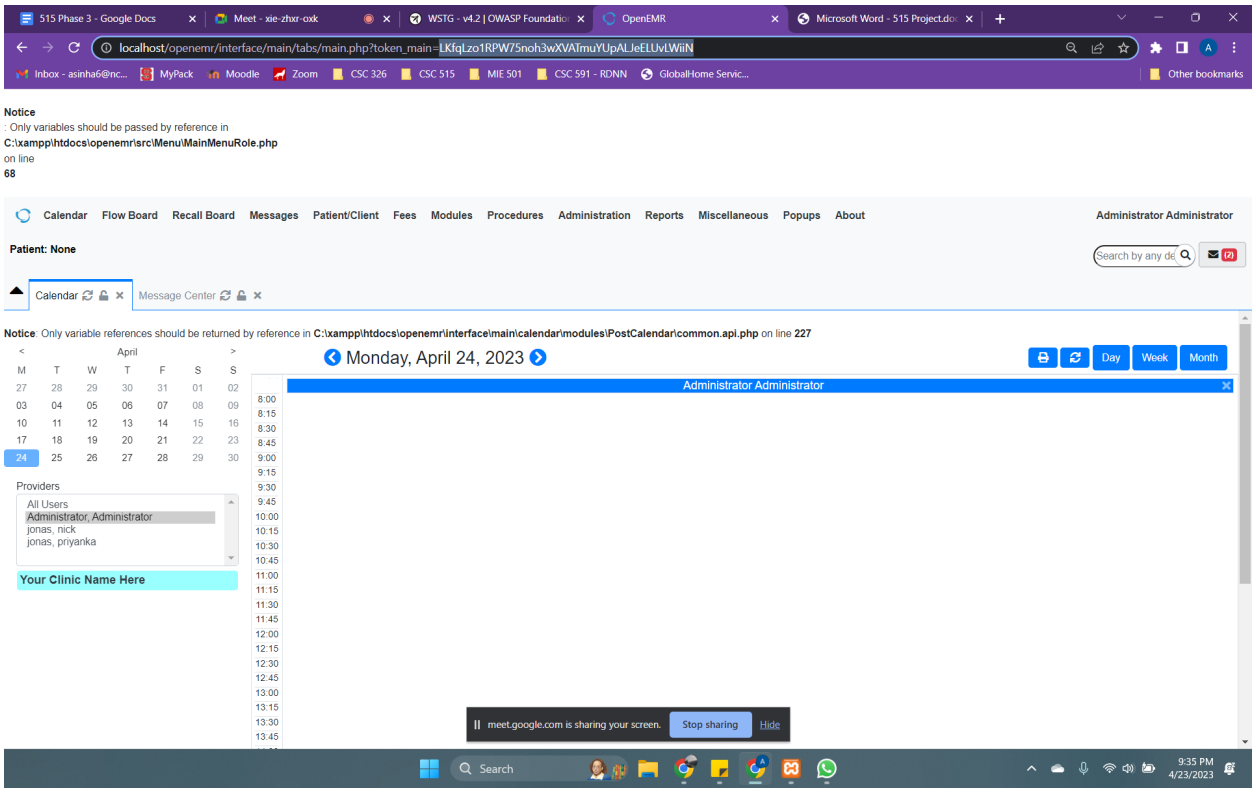
Actual Result: The application properly enforces password change functionality by requiring the user's current and new password, verifying that the new password meets the password policy requirements, and preventing unauthorized access with the old password. The test case passes.



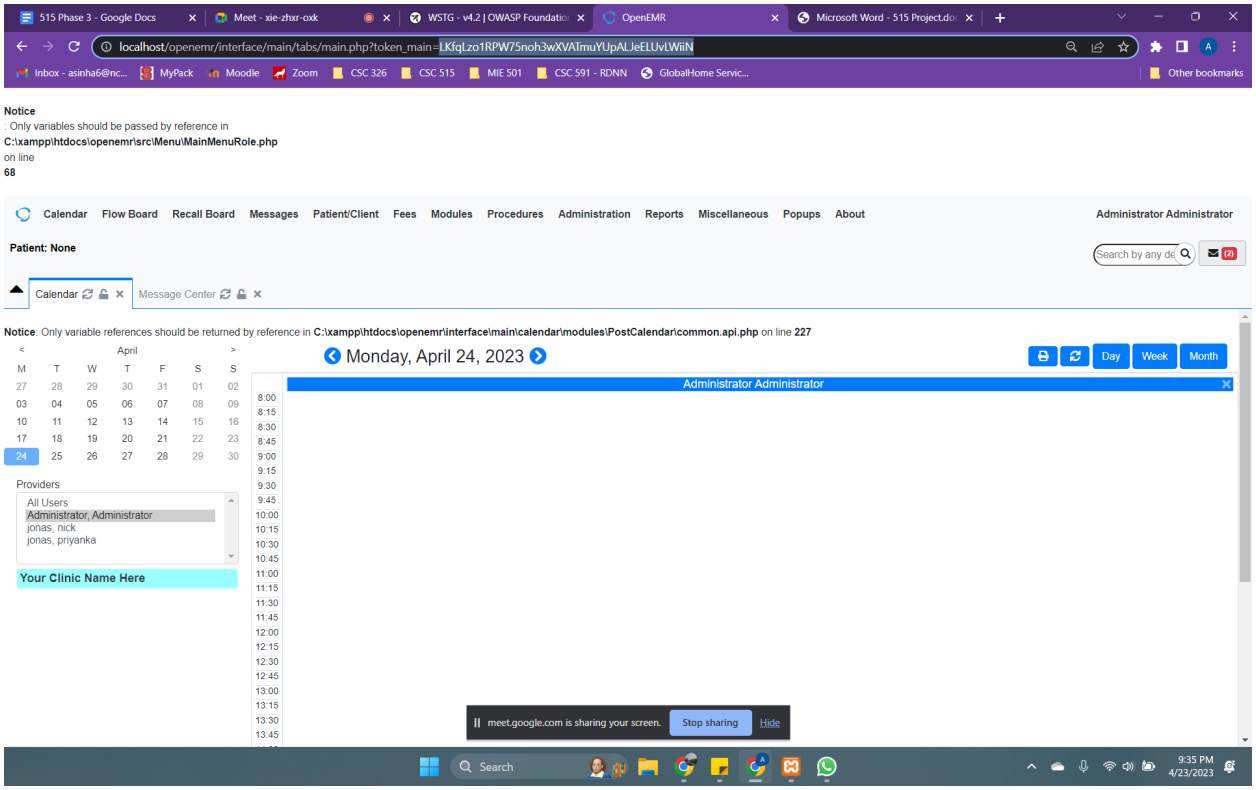
Test Case ID: TC-3.2.1-03:

Actual Result: The application generates a new session token on user authentication. Each user has a unique session token that is not shared between different sessions. The test case passes.

Before logging out:

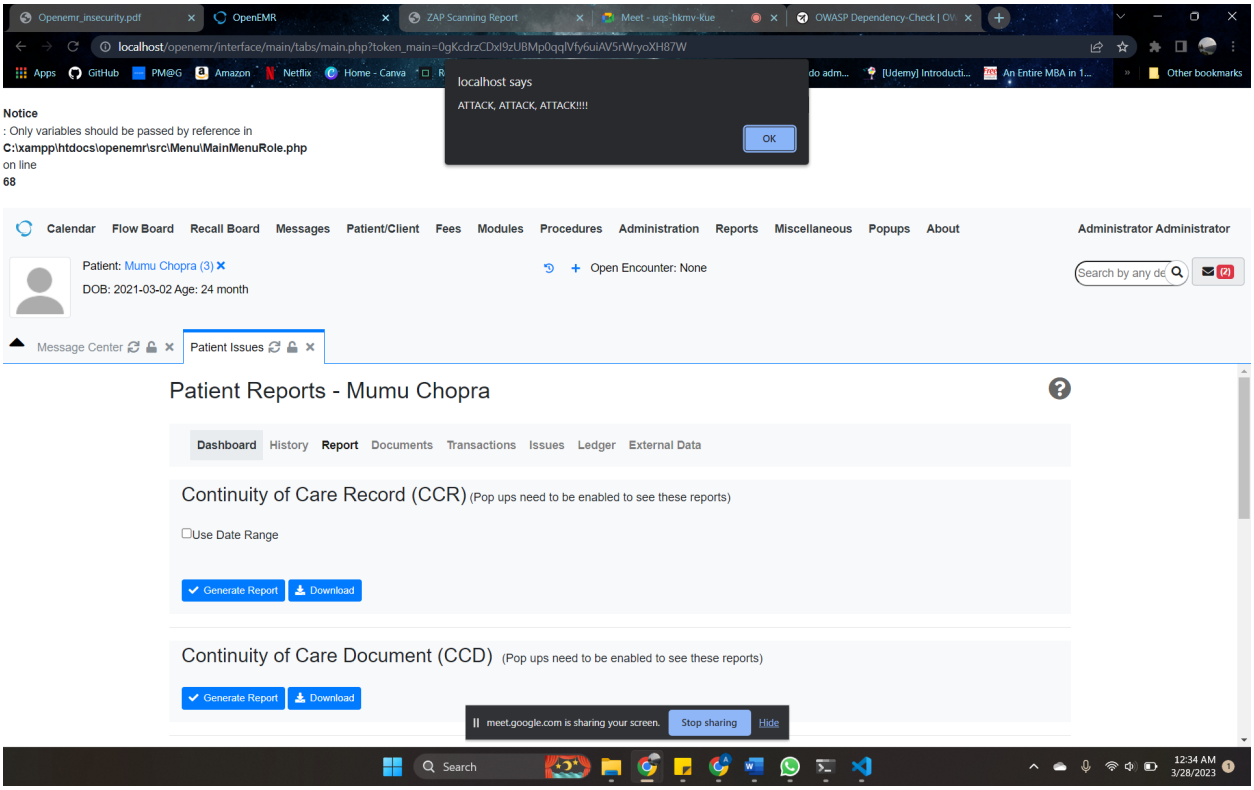


After logging in again:



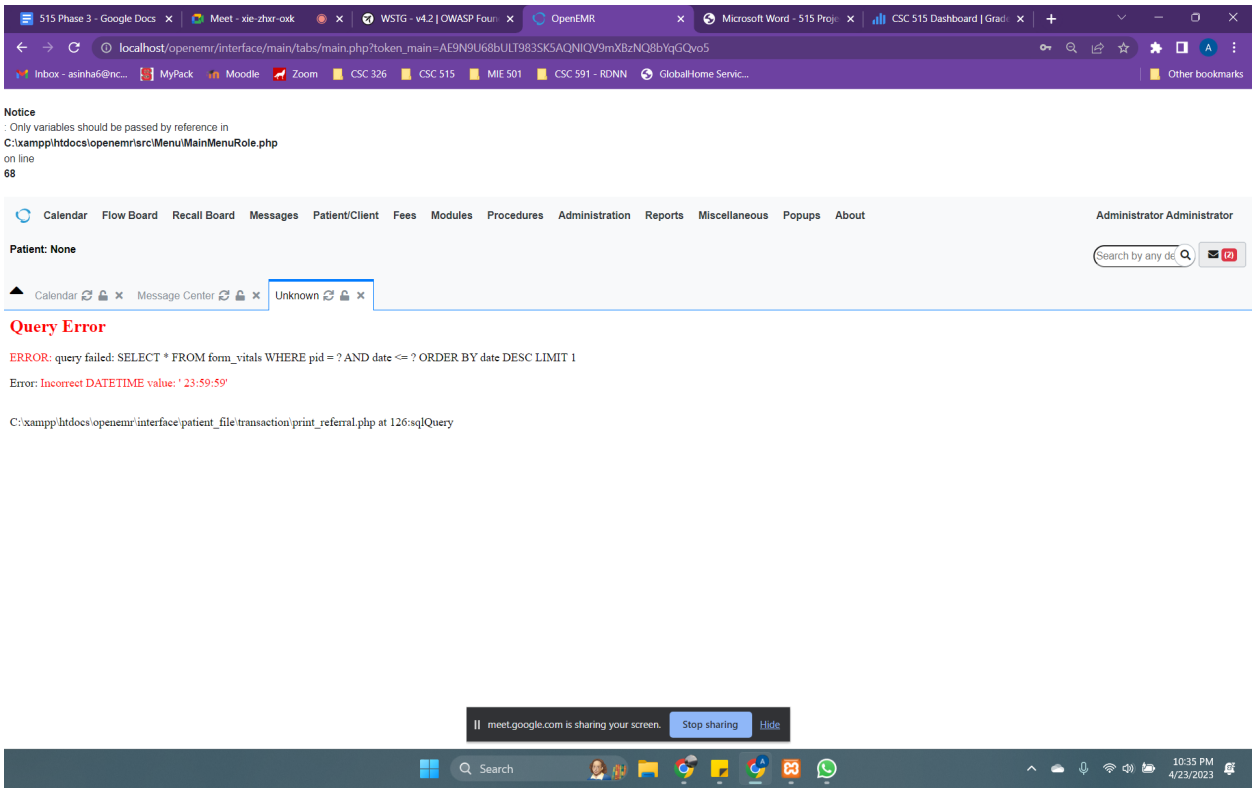
Test Case ID: TC-5.2.8-04

Actual Result: OpenEMR does not sanitize, disable, or sandbox user-supplied scriptable or expression template language content. The application allows untrusted user-supplied content to execute code or perform other malicious activities. The test case fails.



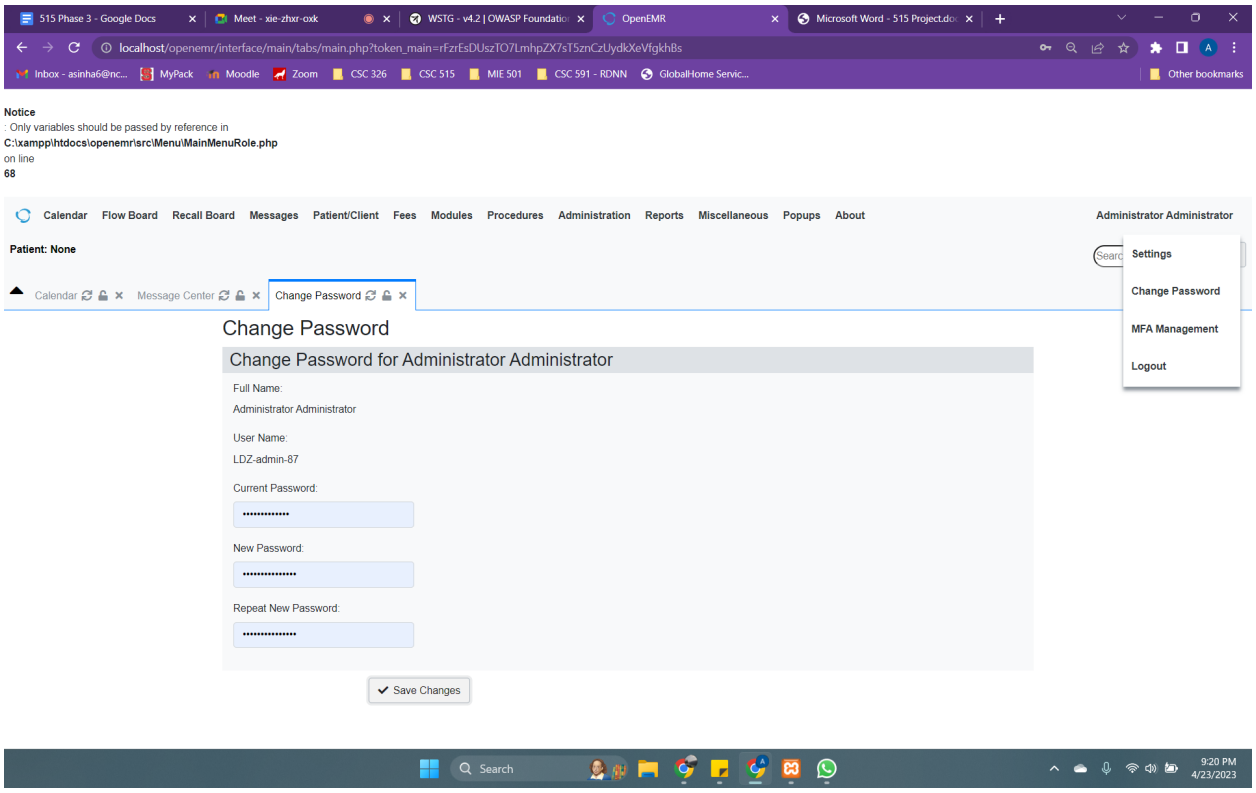
Test Case ID: TC-2.1.8-05

Actual Result: A generic error message is not displayed with a unique ID that support personnel can use to investigate. The error message reveals sensitive information about the application, server, or system. The test case fails.



Test Case ID: TC-2.1.8-06

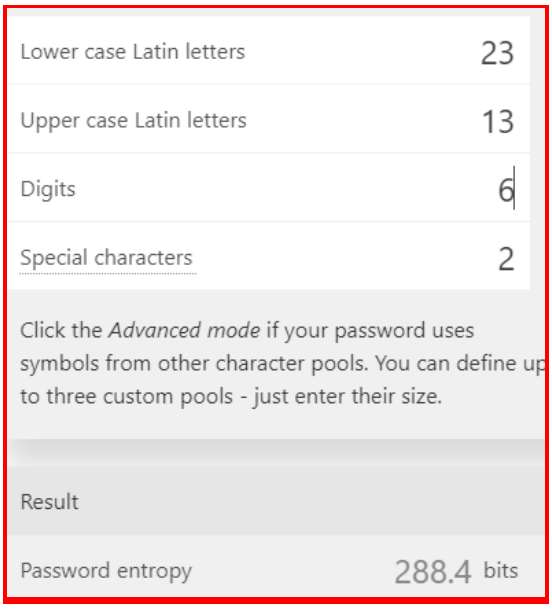
Actual Result: The password strength meter isn't displayed and functional. The strength meter does not provide accurate feedback for passwords of varying strengths. The test case fails.



Test Case ID: TC-2.1.8-07

Actual Result: The session token is included in the HTTP request headers and the session token has at least 64 bits of entropy. The test case passes.

Cookie: OpenEMR=**mjtvMTE1JSenbxuKyAWG6RebFBwjiw1l5F-pHabqOmIMi9gI**; welcomebanner_status=dismiss; language=en; cookieconsent_status=dismiss; continueCode=QVLdwkhOt9UluXteTvfzSmu1iQHYpTPmfL1H85cq2TL6Co4sa9iqoh96dR3w



Test Case ID: TC-2.1.8-08

Actual Result: The Referrer-Policy header is set to an appropriate value such as "strict-origin-when-cross-origin" or "same-origin". The test case passes.