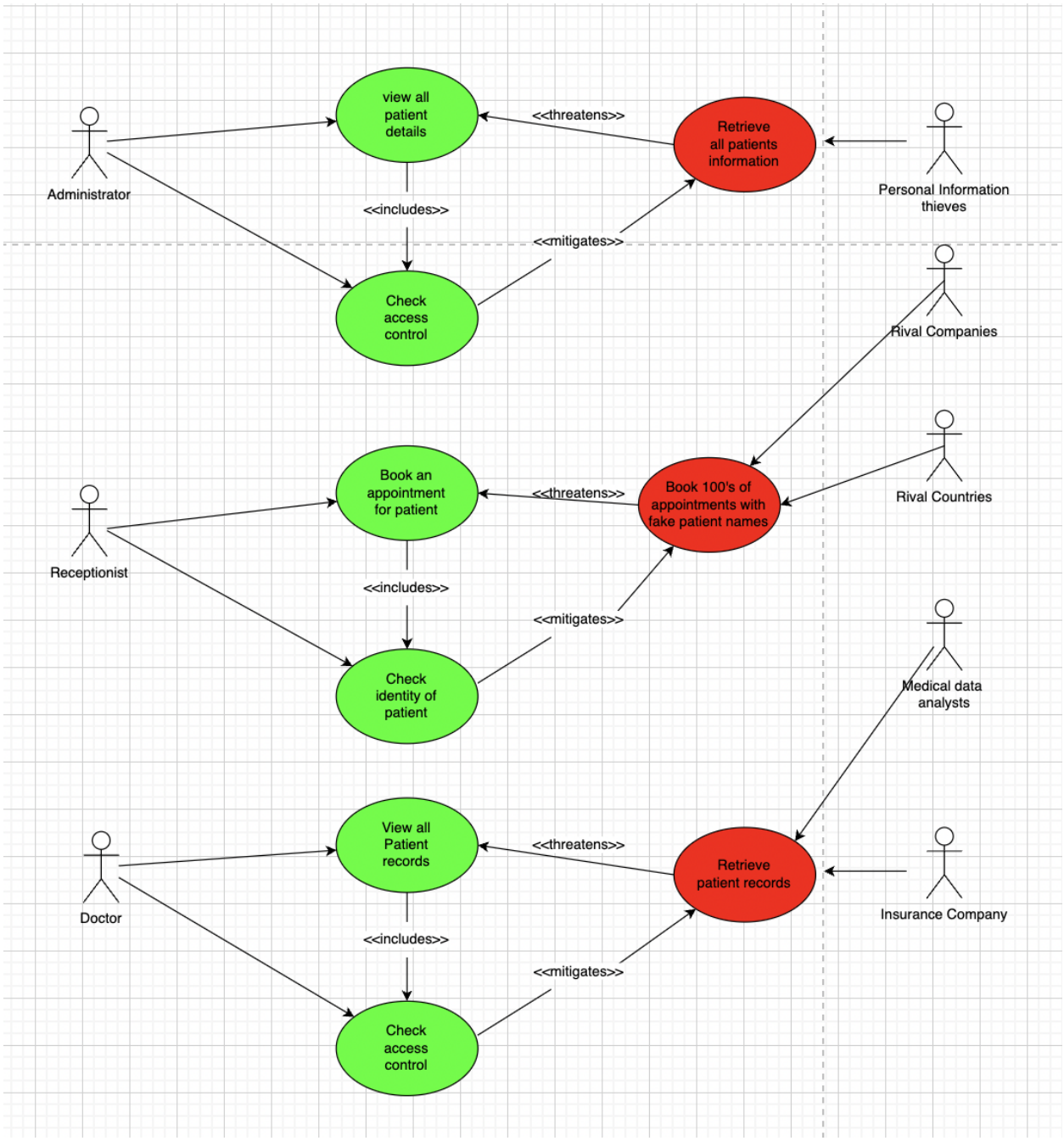
1. Abuse and Misuse Cases

Adversary Types:

1. Personal Information Thieves
2. Rival Companies
3. Rival Countries
4. Medical Data Analysts
5. Insurance Company



Abuse Case Description:
Name: Retrieve Patient Records
Summary: An insurance company retrieves patient records and charges high rate insurance policies for the patients who have severe health conditions.
Date: 04/18/2022
Basic Path: The insurance company hacks the system and login as admin. The hacker from the insurance company goes to the patients tab in openEMR and sees all the patient records and gets the details of patients who are suffering from severe health conditions.
Alternative Path:

- The insurance company has the admin username and password. They don't need to hack the system to view patient records.
- The insurance company uses brute force to crack the admin password.

Capture Points:

- The admin password is changed once every 2 months.
- The password cannot be cracked because it followed standard password requirements.

Preconditions:

- The system has a special user 'admin' with extended privileges.
- The system allows the admin to view the patient records.

Assumptions:

- Only the admin can see all the patient records.

Capture Guarantee (postcondition):

- The insurance company never gets admin access on the OpenEMR

Potential Abuser profile:

- Highly skilled security expert who can hack the openEMR website.
- Highly talented security expert who can crack the admin password based on his personal information.

---

2. Security Requirements

S.No.	Control Number	User Story	Acceptance Criteria	Source
1	ASVS - 2.2.1	As an OpenEMR user, when I try to login with a wrong password more than 3 times, my account should be locked out for more than 24 hours.	Authenticate the user for the correctness of the password. If it is incorrect more than 3 times, that specific user should be locked out for more than 24 hours.	Threat Model
2	ASVS - 5.1.3	As an OpenEMR user, when I try to create a new patient no malicious characters should be accepted as part of	Malicious characters like **<$%** should not be accepted as part of a patient name. Allow lists should be used to block those kinds of	Threat Model

		the patient's name.	characters.	
3	ASVS - 4.3.2	As an OpenEMR User, only privileged accounts should have access to sensitive files.	Only files authorized according to the business use case should be able to be accessed by the user. Apart from those, the user should access no other files.	Threat Model
4	ASVS - 2.2.4	As an OpenEMR User, when I login as a privileged user, multi-factor Authentication should be enabled.	Use two or more pieces of evidence to authenticate a user such as username and password in addition to a token from a physical smart card or token generator.	Attack/Defens e Trees
5	ASVS - 9.1.1	As an OpenEMR User, I want all communication between the application and all clients to be done using TLS(https)	All the communications between clients, and within the application from (frontend and backend) should use HTTPS. So, the risk of a MITM attack is lowered.	Attack/Defens e Trees
6	ASVS - 2.1.1	As an OpenEMR developer, I want to ensure that Passwords are strong enough so that no attacker can crack the password to steal the valuable information.	OpenEMR adversaries like insurance companies should not be able to crack admin password, to ensure the safety of patient health records.	Misuse/abuse cases
7	ASVS - 2.5.7	As an OpenEMR developer, I want the patient to be authenticated before booking an appointment so that the fake appointments can be avoided.	OpenEMR adversaries like rival companies or rival countries should not be able to book appointments with fake patient details which can affect the OpenEMR business.	Misuse/abuse cases
8	ASVS - 2.10.4	As an OpenEMR User, I should not be able to access the API keys, and passwords from the source code, or online source code repositories.	The source code should contain only the information using which an attacker should not be able exploit the application and gain access to the data in the backend.	Threat Model
9	NIST 800-53 - AC-2(3)	As an OpenEMR User, My account should be disabled when it has expired, or is no longer associated with me, or in violation of organization policy, or been inactive for organization-defined time	The expired, inactive, or otherwise anomalous accounts should be disabled to support the concepts of least privilege and least functionality which reduce the attack surface of the system.	Policy or standards (HIPAA)

		periods.		
10	NIST 800-53 - AC-2(5)	As an OpenEMR User, my account should be logged out if I am temporarily inactive for longer than my organization's defined time period.	The temporarily inactive account should be automatically logged out after the organization-defined time period to reduce the risk of an attacker physically using a logged in machine.	Policy or standards (HIPAA)

---

3. Vulnerability Fixes

1. Sanitizing the input fields in a form.

Reference: Project 1, Black box test case 1.

10. V5.2 Sanitization and Sandboxing Requirements

Test Case ID: 5.2.2-1

Description: Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.

Repeatable Step:

1. Login as any user (Eg: username: admin_openemr password: admin_openemr)
2. Under the administration tab, click on Users
3. Click on the username (Eg: admin_openemr)
4. Enter the following for the last name:
    - id=<iframe src="javascript:alert(document.cookie)">
5. Log out and log back in to view the user's name in the top right corner

Expected Results:

- The input should be sanitized to allow only letters and whitespace.

Black box test case (Clear steps):

Test Case ID: 5.2.2-1

Description: Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.

Repeatable Steps:

1. Login to vcl and start the openemr application.
2. Login as admin. (username: admin_openemr password: admin_openemr)
3. Under the administration tab, click on Users

4. Click on the username (Eg: admin_openemr)
5. Enter the following for the first name:
    - <iframe src="javascript:alert(document.cookie)">
6. Log out and log back in to view the user's name in the top right corner

Expected Results: The input should be sanitized to allow only letters and whitespace.

CWE Info: 138: Improper Neutralization of Special Elements

Code Fix: For this vulnerability, all the input fields in the editable form are sanitized, such that no malicious characters get saved into db.

Pull Request: https://github.com/openemr/openemr/pull/5172

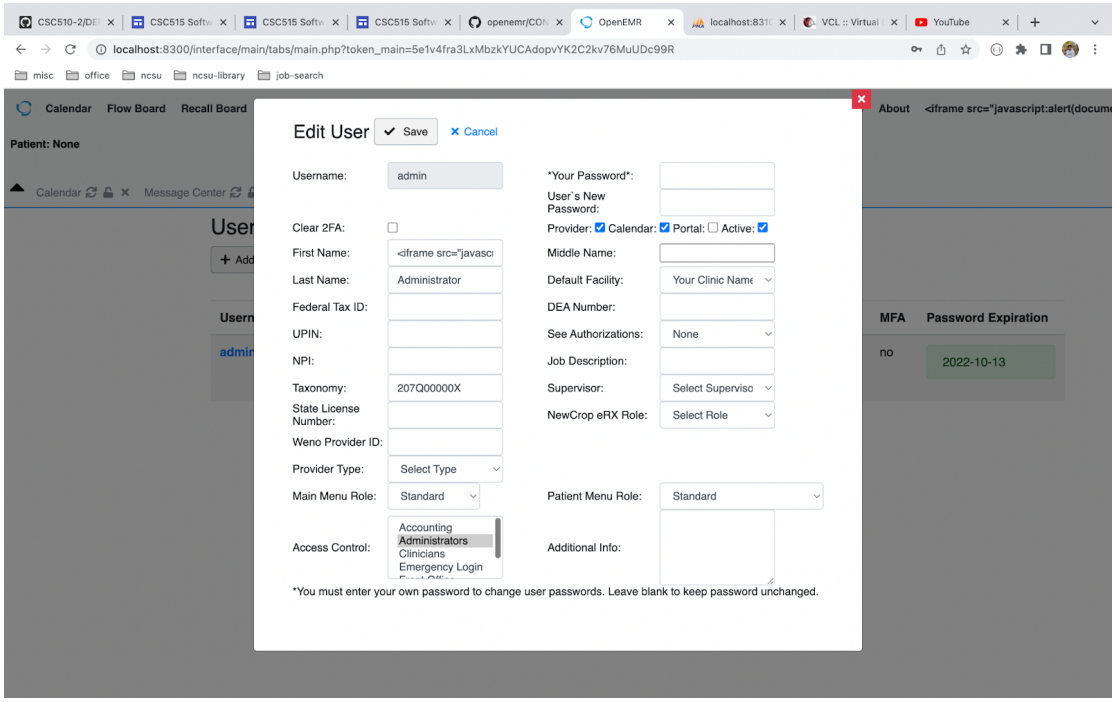
22 interface/usergroup/usergroup_admin.php

```
@@ -112,39 +112,39 @@
        }

        if ($_POST["taxid"]) {
-           sqlStatement("update users set federaltaxid=? where id= ? ", array($_POST["taxid"], $_POST["id"]));
+           sqlStatement("update users set federaltaxid=? where id= ? ", array(filter_var ( $_POST["taxid"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["state_license_number"]) {
-           sqlStatement("update users set state_license_number=? where id= ? ", array($_POST["state_license_number"], $_POST["id"]));
+           sqlStatement("update users set state_license_number=? where id= ? ", array(filter_var ( $_POST["state_license_number"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["drugid"]) {
-           sqlStatement("update users set federaldrugid=? where id= ? ", array($_POST["drugid"], $_POST["id"]));
+           sqlStatement("update users set federaldrugid=? where id= ? ", array(filter_var ( $_POST["drugid"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["upin"]) {
-           sqlStatement("update users set upin=? where id= ? ", array($_POST["upin"], $_POST["id"]));
+           sqlStatement("update users set upin=? where id= ? ", array(filter_var ( $_POST["upin"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["npi"]) {
-           sqlStatement("update users set npi=? where id= ? ", array($_POST["npi"], $_POST["id"]));
+           sqlStatement("update users set npi=? where id= ? ", array(filter_var ( $_POST["npi"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["lname"]) {
-           sqlStatement("update users set lname=? where id= ? ", array($_POST["lname"], $_POST["id"]));
+           sqlStatement("update users set lname=? where id= ? ", array(filter_var ( $_POST["lname"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["job"]) {
-           sqlStatement("update users set specialty=? where id= ? ", array($_POST["job"], $_POST["id"]));
+           sqlStatement("update users set specialty=? where id= ? ", array(filter_var ( $_POST["job"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["mname"]) {
-           sqlStatement("update users set mname=? where id= ? ", array($_POST["mname"], $_POST["id"]));
+           sqlStatement("update users set mname=? where id= ? ", array(filter_var ( $_POST["mname"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if ($_POST["facility_id"]) {
@@ -219,7 +219,7 @@
        }

        if ($_POST["fname"]) {
-           sqlStatement("update users set fname=? where id= ? ", array($_POST["fname"], $_POST["id"]));
+           sqlStatement("update users set fname=? where id= ? ", array(filter_var ( $_POST["fname"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }

        if (isset($_POST['default_warehouse'])) {
@@ -270,7 +270,7 @@
        }

        if ($_POST["comments"]) {
-           sqlStatement("update users set info = ? where id = ? ", array($_POST["comments"], $_POST["id"]));
+           sqlStatement("update users set info = ? where id = ? ", array(filter_var ( $_POST["comments"], FILTER_SANITIZE_STRING), $_POST["id"]));
        }
```

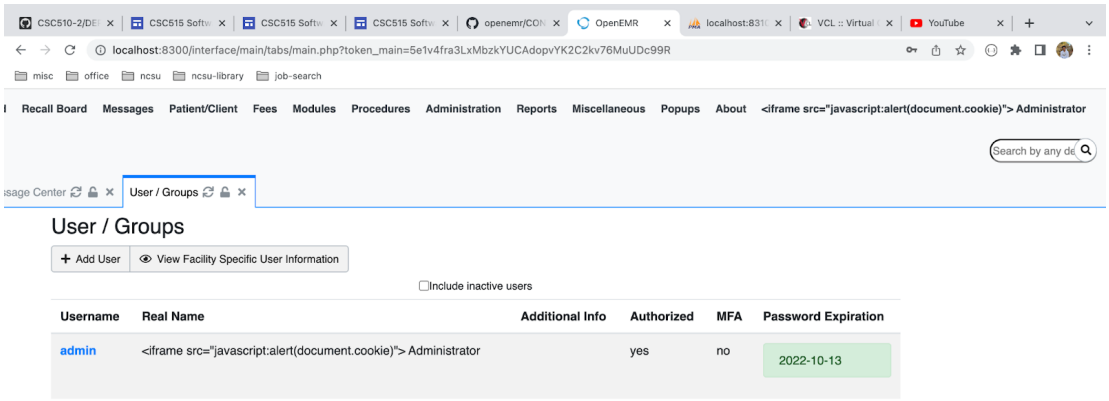
```
292
293         if ($_POST["erxprid"]) {
294 -           sqlStatement("update users set weno_prov_id = ? where id = ? ",
    array($_POST["erxprid"], $_POST["id"]));
295         }
296
297         if (isset($_POST["supervisor_id"])) {
```

```
292
293         if ($_POST["erxprid"]) {
294 +           sqlStatement("update users set weno_prov_id = ? where id = ? ",
    array(filter_var ( $_POST["erxprid"], FILTER_SANITIZE_STRING), $_POST["id"]));
295         }
296
297         if (isset($_POST["supervisor_id"])) {
```

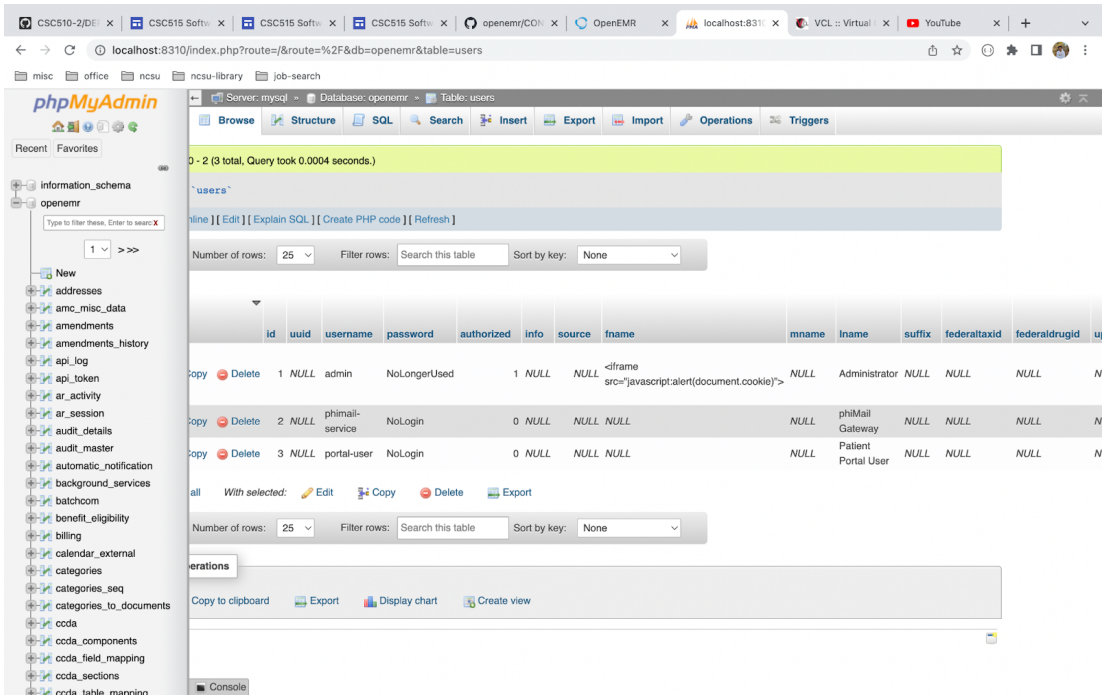
Screenshots before code fix:



In the Edit user form, First Name is replaced with **<iframe src="javascript:alert(document.cookie)">** and clicked on save.

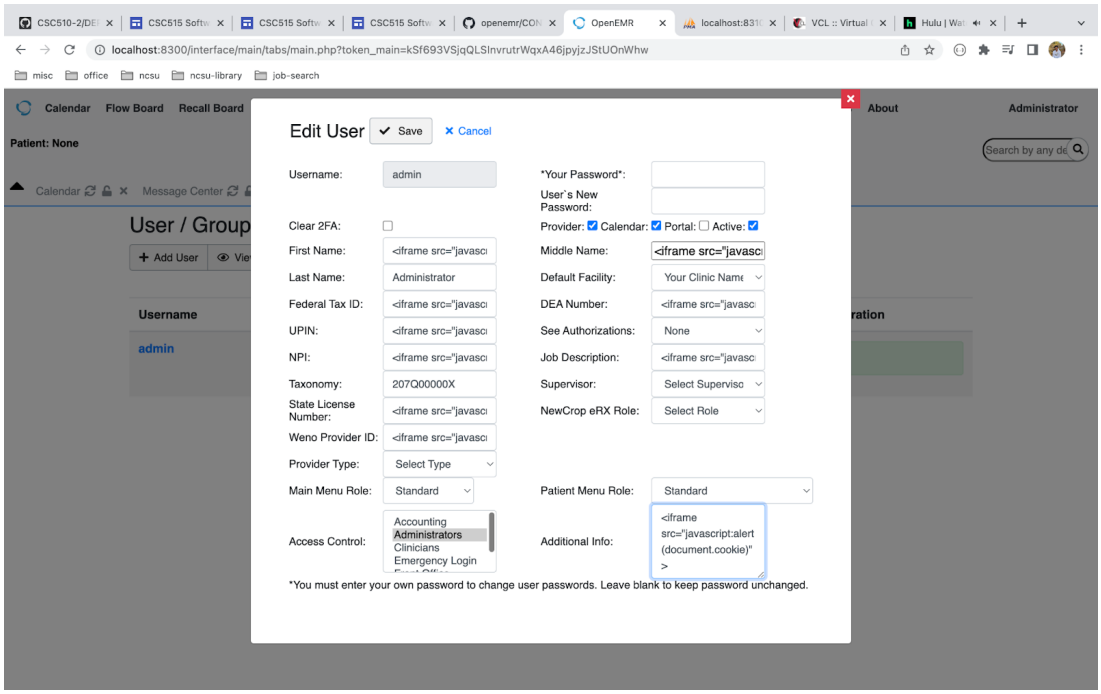


The executable script is injected and is being displayed on the screen without sanitization.

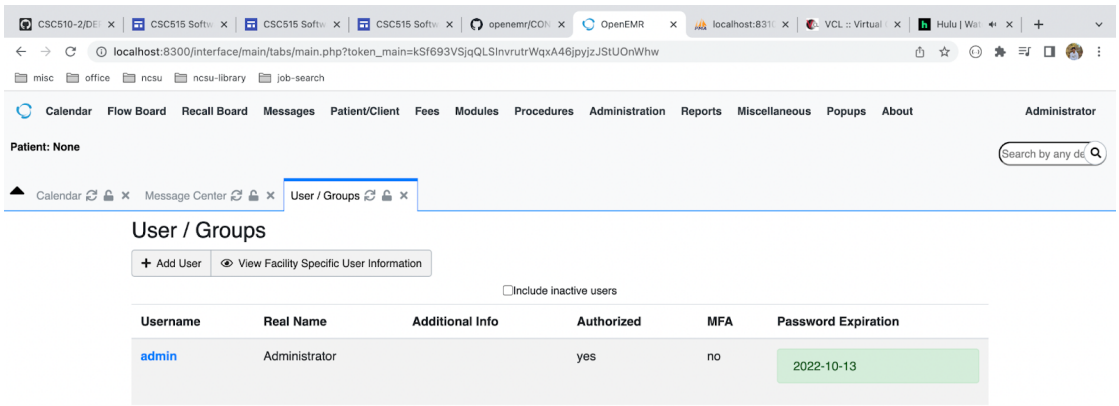


The executable script is being saved in the database without sanitization.

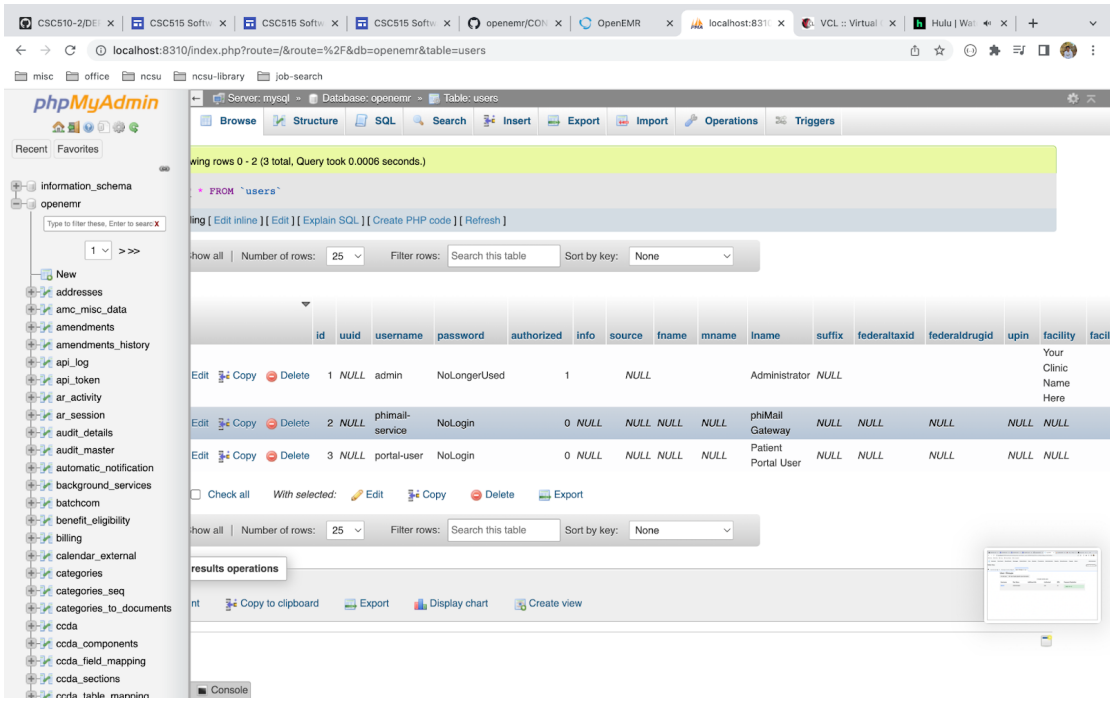
Screenshots after code fix:



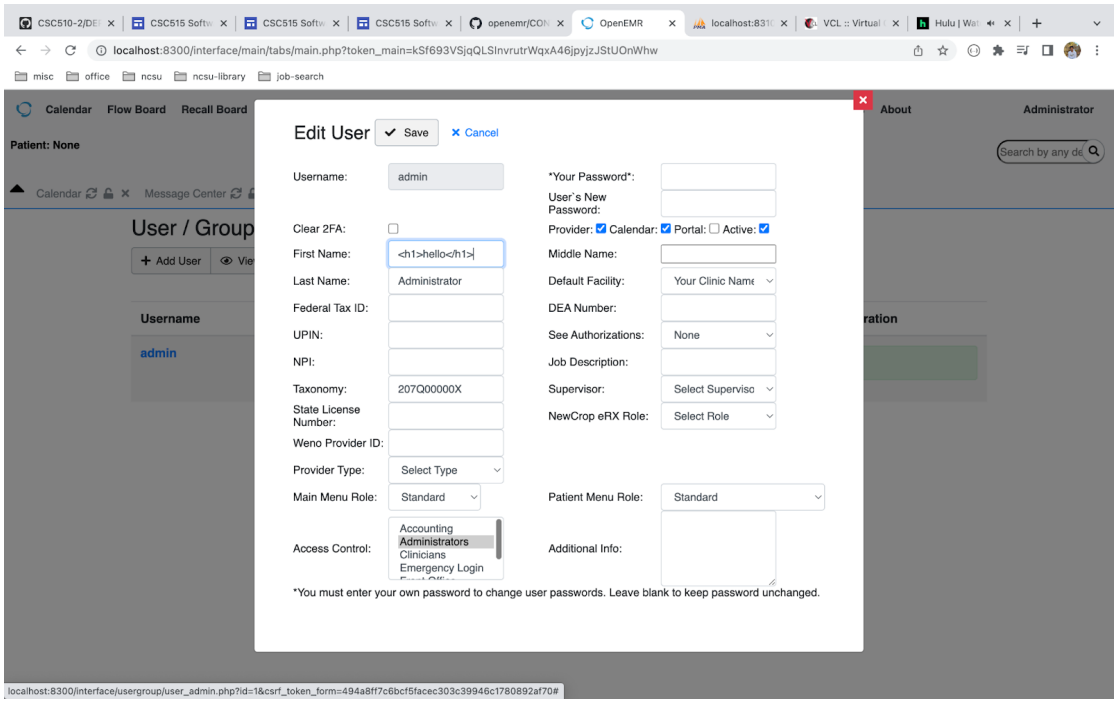
Once the code fix is done, First Name is replaced with **<iframe src="javascript:alert(document.cookie)">** and clicked on save.



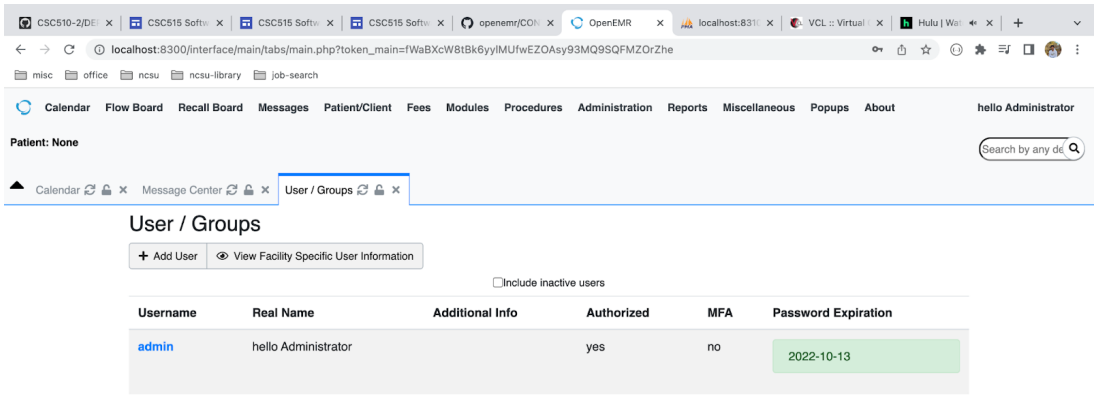
Since the text in the first name contains malicious characters, it’s sanitized. All the characters between < and > will be deleted.



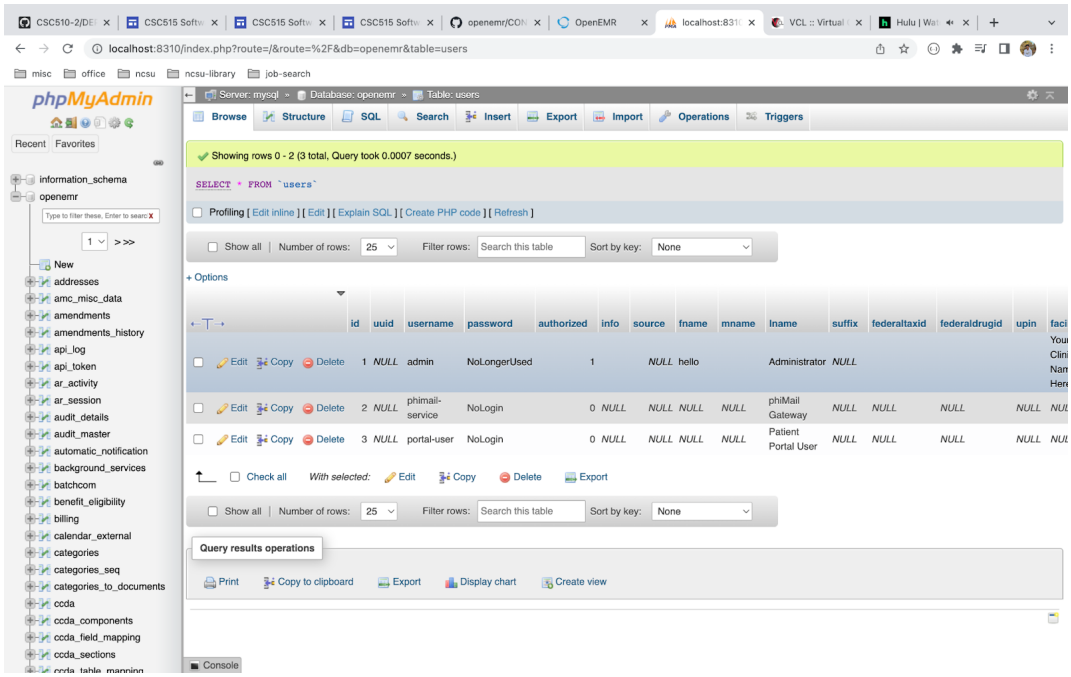
Since all the text in fname is between < and > no text is saved. An empty string is saved as fname.



Now the First Name is replaced with **<h1>hello</h1>** and clicked on save.



The tags from the input <h1>hello</h1> are sanitized and only hello gets saved as a first name.



The tags from the input <h1>hello</h1> are sanitized and only hello got saved as fname in the database.

2. Adding Upload File Restrictions (Newly Created Test Case)

12.2. File Integrity

Test Case ID: 12.2.1-1

Description: Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.

Repeatable Step:

1. Login as any user (Eg: username: admin_openemr password: admin_openemr)
2. Under the **Fees** tab, hover the mouse over **EDI History and click on it.**
3. Select "**Choose File**" under "**New files**".
4. Choose a file type such as exe, .bat
5. Click on submit button below to make an upload API request to process the file.
6. Also, click on the EDI file tab under EDI history tab to repeat the steps 4 and 5.

Expected Results:

- The system should restrict the uploading of dangerous file types with front end validations incorporated.

CWE Info: 434: Unrestricted Upload of File with Dangerous Type

Tool Used to find: Web Inspect

WebInspect Report Screenshot:

Often Misused: File Upload (909) View Description

CWE: 434

Kingdom: API Abuse

Page: http://152.7.177.250:80/openemr/interface/billing/edi_271.php

Request:

```
GET /openemr/interface/billing/edi_271.php HTTP/1.1
Referer: http://152.7.177.250/openemr/interface/billing/
Accept: */*
Pragma: no-cache
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0)
Gecko/20100101 Firefox/83.0
Host: 152.7.177.250
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.103
X-Scan-Memo:
Category="Crawl";SID="59E36790F1EAA5436B910BC44A905C4C";PSID="5047E8FAB8CA7B
D8C7D1E4A7AA2332CC";SessionType="Crawl";CrawlType="HTML";AttackType="None";O
riginatingEngineID="00000000-0000-0000-0000-
000000000000";AttributeName="href";Format="NonRooted";LinkKind="HyperLink";L
ocations="HtmlNode";Source="ScriptExecution";tht="31";
X-RequestManager-Memo: stid="54";stmi="0";sc="1";rid="4303bb4d";
X-Request-Memo: rid="c8f7b8fa";sc="2";thid="76";
Cookie:
CustomCookie=WebInspect175190ZX2CB257C069934F78807027414B0AAD7DYDB97;OpenEMR
=jETu3s3ZOkm3j%2Cc3f37jiR2%2CT8Ow04w9GKaHm-FX1gT%
2CCdET;OpenEMR=jETu3s3ZOkm3j%2Cc3f37jiR2%2CT8Ow04w9GKaHm-FX1gT%2CCdET
```

Before Fix:

As shown in the below screenshot, any file type for e.g: exe and .bat files can be uploaded as an edi file, as there is no front end validation in place.

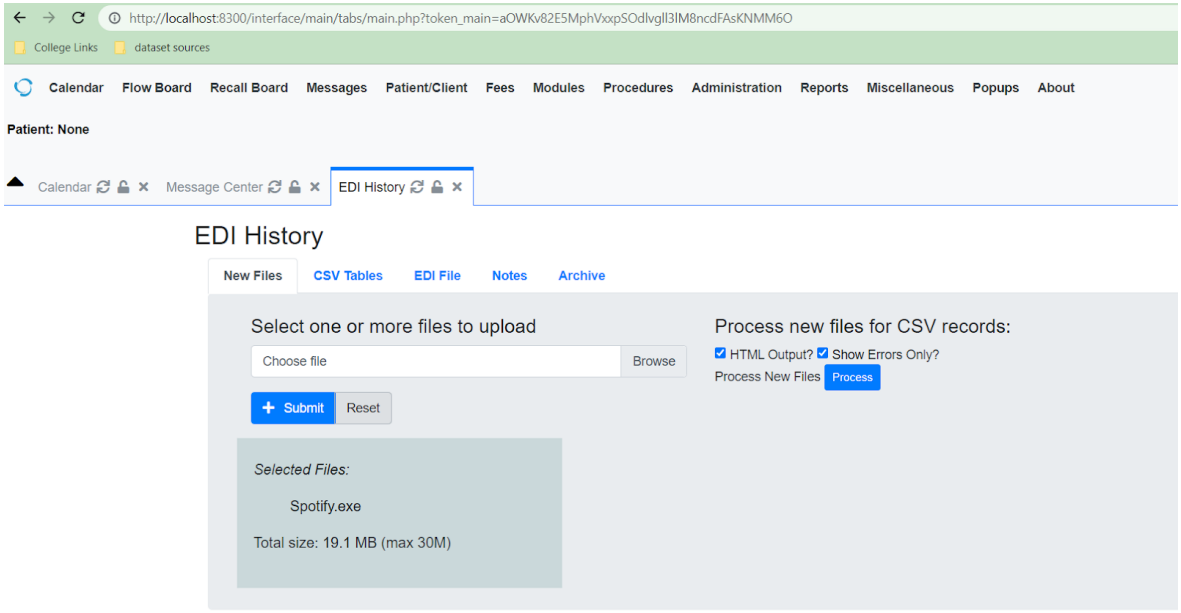
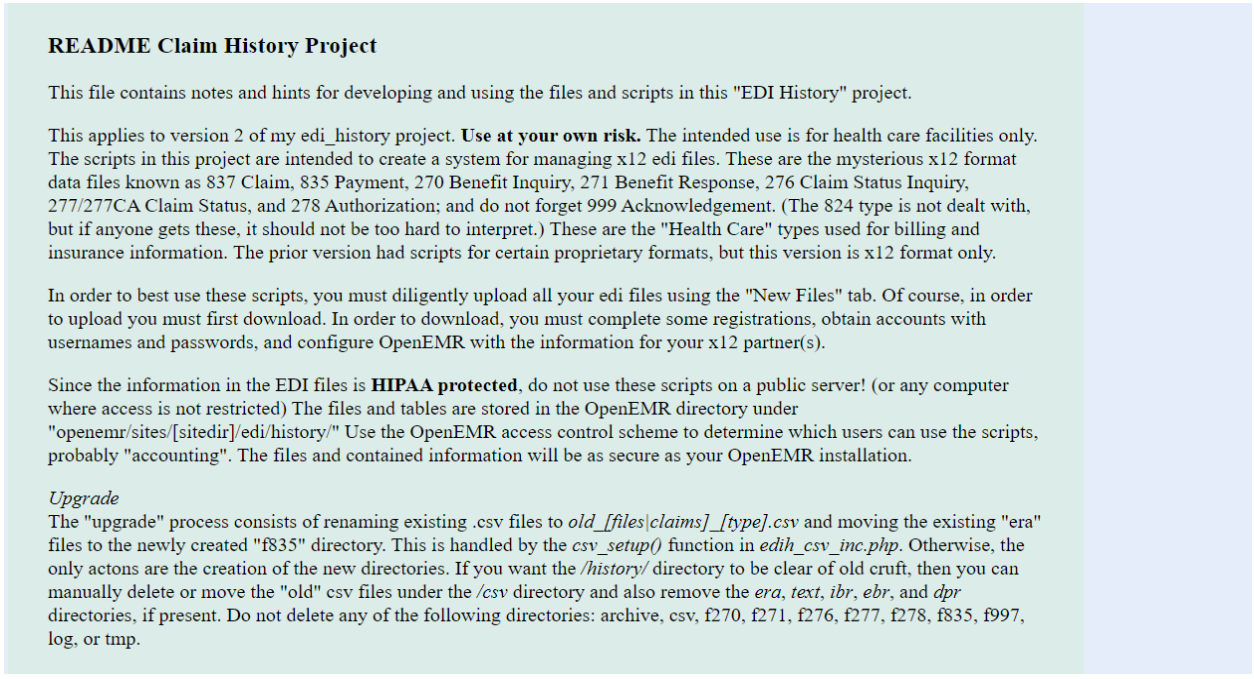


Fig 1



README Claim History Project

This file contains notes and hints for developing and using the files and scripts in this "EDI History" project.

This applies to version 2 of my edi_history project. **Use at your own risk.** The intended use is for health care facilities only. The scripts in this project are intended to create a system for managing x12 edi files. These are the mysterious x12 format data files known as 837 Claim, 835 Payment, 270 Benefit Inquiry, 271 Benefit Response, 276 Claim Status Inquiry, 277/277CA Claim Status, and 278 Authorization; and do not forget 999 Acknowledgement. (The 824 type is not dealt with, but if anyone gets these, it should not be too hard to interpret.) These are the "Health Care" types used for billing and insurance information. The prior version had scripts for certain proprietary formats, but this version is x12 format only.

In order to best use these scripts, you must diligently upload all your edi files using the "New Files" tab. Of course, in order to upload you must first download. In order to download, you must complete some registrations, obtain accounts with usernames and passwords, and configure OpenEMR with the information for your x12 partner(s).

Since the information in the EDI files is **HIPAA protected**, do not use these scripts on a public server! (or any computer where access is not restricted) The files and tables are stored in the OpenEMR directory under "openemr/sites/[sitedir]/edi/history/" Use the OpenEMR access control scheme to determine which users can use the scripts, probably "accounting". The files and contained information will be as secure as your OpenEMR installation.

Upgrade
The "upgrade" process consists of renaming existing .csv files to *old_[files|claims]_[type].csv* and moving the existing "era" files to the newly created "f835" directory. This is handled by the *csv_setup()* function in *edih_csv_inc.php*. Otherwise, the only actons are the creation of the new directories. If you want the */history/* directory to be clear of old cruft, then you can manually delete or move the "old" csv files under the */csv* directory and also remove the *era*, *text*, *ibr*, *ebr*, and *dpr* directories, if present. Do not delete any of the following directories: archive, csv, f270, f271, f276, f277, f278, f835, f997, log, or tmp.

Fig 2

```
lass="container">
v class="row">
 <div class="col-sm-12 col-md-6">
    <form id="formupl" name="form_upl" action="edih_main.php" method="POST" enctype="multipart/form-data">
       <input type="hidden" name="csrf_token_form" value="<?php echo attr(CsrfUtils::collectCsrfToken()); ?>" />
       <h4><?php echo xlt("Select one or more files to upload"); ?></h4>
       <div class="custom-file">
          <label class="custom-file-label"><?php echo xlt("Choose file"); ?></label>
          <input type="file" class="custom-file-input" id="uplmulti" name="fileUplMulti[]" multiple />
          <input type="hidden" name="NewFiles" form="formupl" value="ProcessNew" />
          <div class="btn-group mt-3">
             <button type="submit" class="btn btn-primary btn-add" id="uplsubmit" name="upl_submit" form="formupl" value="<?php echo
                <?php echo xlt("Submit"); ?>
             </button>
             <input type="reset" class="btn btn-secondary" id="uplreset" name="upl_reset" form="formupl" value="<?php echo xla("Rese
          </div>
       </div>
    </form>
```

Fig 3

```
    <td class='text-left'>
       <div class="custom-file">
          <label class="custom-file-label"><?php echo xlt("Choose file"); ?></label>
          <input id="x12file" type="file" class="custom-file-input" size=30 name="fileUplx12" accept=".x12,.csv,.zip,.txt,.edi" />
       </div>
    </td>
```

Fig 4

According to the edi project, any file with x12, zip, txt and csv files can be uploaded to be further processed for the potential data transfer between the partners. So, only such file types should be allowed and the rest of the other file types should be restricted.

Also, as shown in Fig 3 the input element allows all the file types to be uploaded.

Fix provided:

As shown in the below screenshot, the fix has been implemented to only allow the extensions like (.x12, .zip, .txt, .csv)

```
  <input type="hidden" name="csrf_token_form" value="<?php echo attr(CsrfUtils::collectCsrfToken()); ?>" />
  <h4><?php echo xlt("Select one or more files to upload"); ?></h4>
  <div class="custom-file">
     <label class="custom-file-label"><?php echo xlt("Choose file"); ?></label>
     <input type="file" class="custom-file-input" id="uplmulti" name="fileUplMulti[]" accept=".x12,.csv,.zip,.txt,.edi" multiple />
     <input type="hidden" name="NewFiles" form="formupl" value="ProcessNew" />
     <div class="btn-group mt-3">
        <button type="submit" class="btn btn-primary btn-add" id="uplsubmit" name="upl_submit" form="formupl" value="<?php echo xla("S
           <?php echo xlt("Submit"); ?>
        </button>
        <input type="reset" class="btn btn-secondary" id="uplreset" name="upl_reset" form="formupl" value="<?php echo xla("Reset"); ?>
     </div>
  </div>
```

Fig 5

```
</td>
<td class='text-left'>
    <div class="custom-file">
        <label class="custom-file-label"><?php echo xlt("Choose file"); ?></label>
        <input id="x12file" type="file" class="custom-file-input" size=30 name="fileUplx12" accept=".x12,.csv,.zip,.txt,.edi" />
    </div>
</td>
```

Fig 6

As shown in the Fig 5, the input element is not allowed to upload the file types like .exe and .bat files which were able to be uploaded previously.

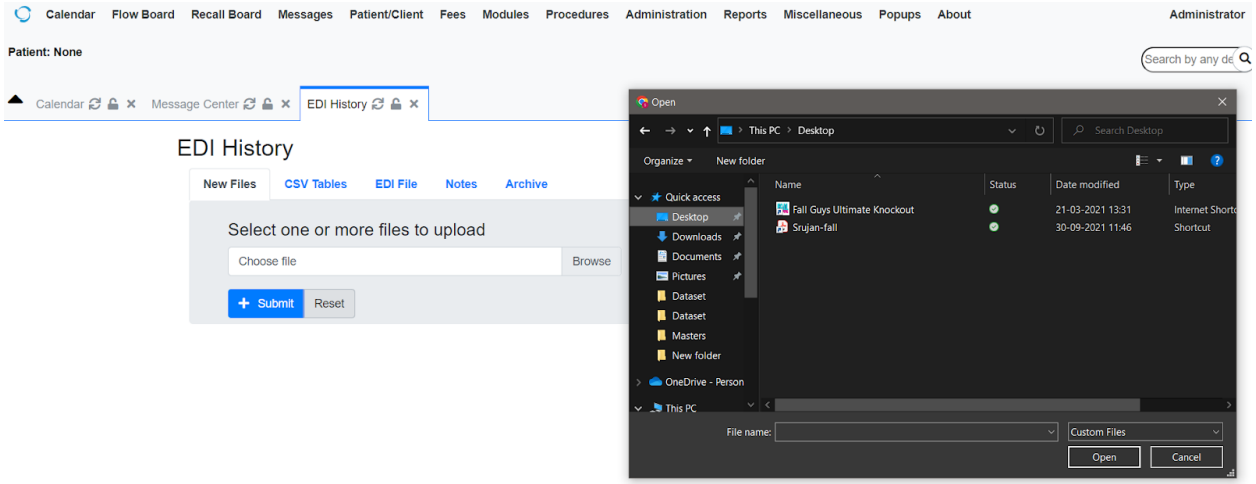


Fig 7

3. Preventing browsers to save password and autocomplete login forms.

Reference: Project 3, Black box test case 4 in test coverage section.

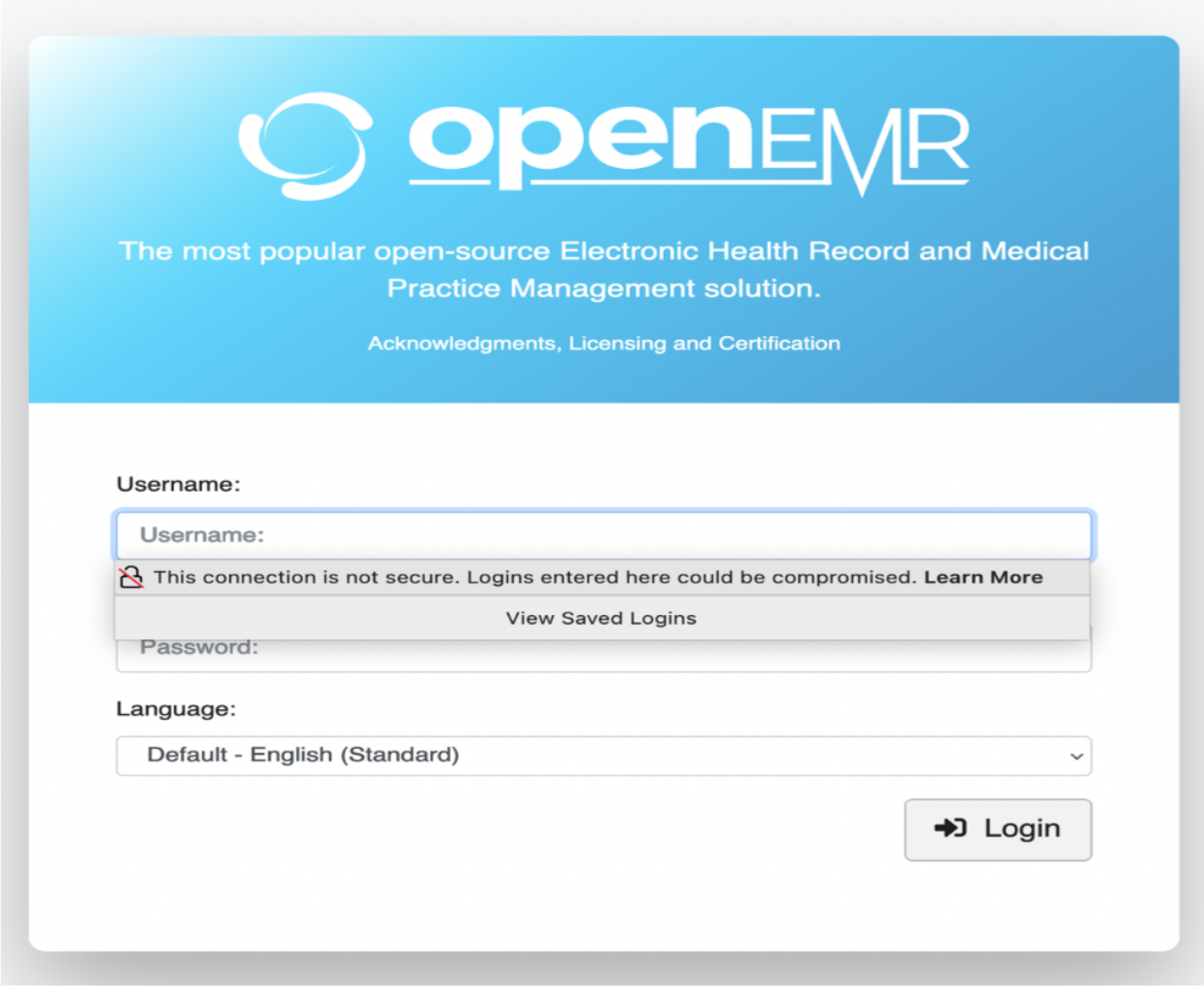
4. Test Case ID: 8.2.1-1
Description: Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.
Repeatable Steps:
1. Open openemr application in one of the following autocomplete enabled browsers.
    Internet Explorer version 11 or above
    Firefox version 30 or above
    Chrome version 34 or above
2. Try login using the credentials. (username: admin_openemr, password: admin_openemr). For example in firefox, during login you will notice some pop ups to save credentials.

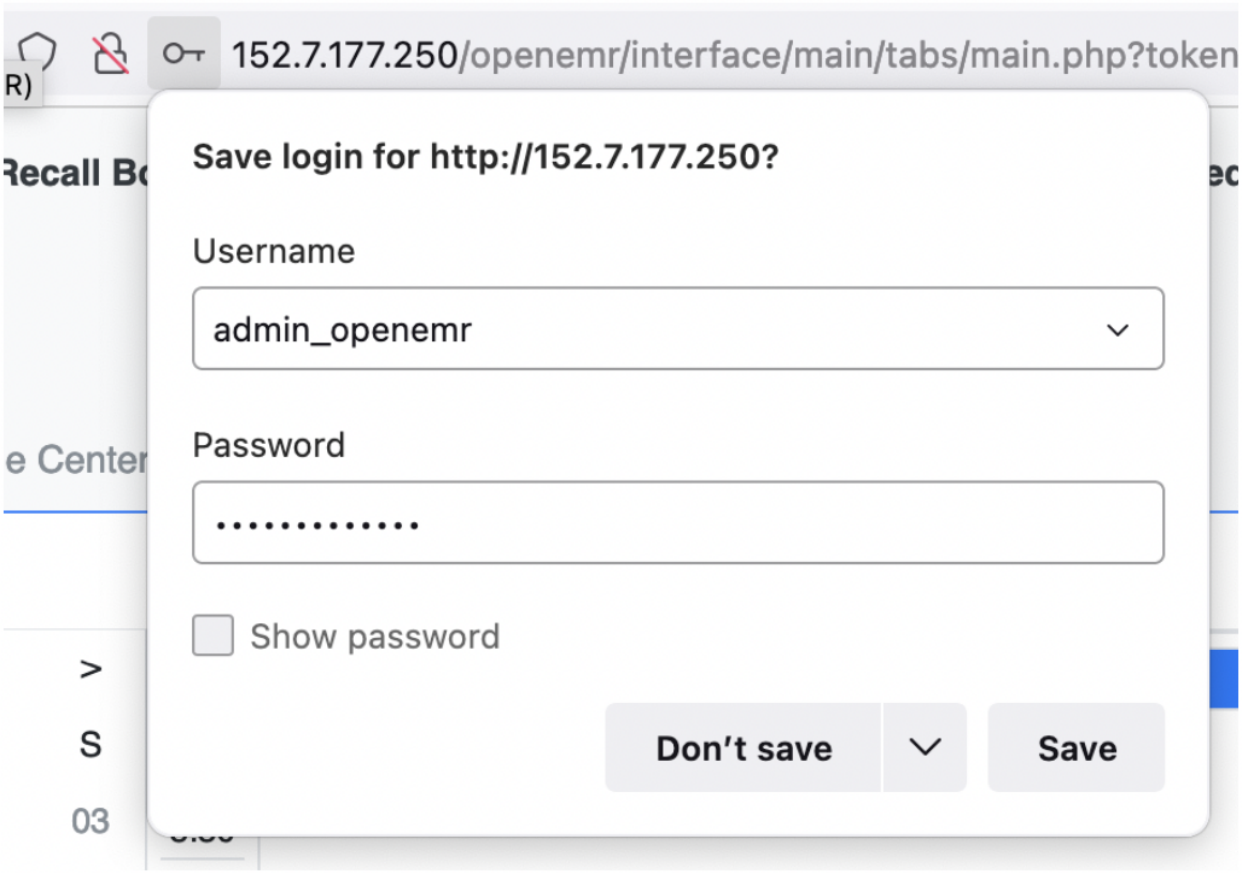
Black box test case (Clear steps):

Test Case ID: 8.2.2-1

Description: Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.

Repeatable Steps:
1. Login to vcl and start the openemr application.
2. Open openemr application in one of the following autocomplete enabled browsers.
    Internet Explorer version 11 or above
    Firefox version 30 or above
    Chrome version 34 or above
3. Try login using the credentials. (username: admin_openemr, password: admin_openemr). For example in firefox, during login you will notice some pop ups to save credentials.





4. Click on save and now try to re-login using saved password credentials in the browser.

Expected Results: When autocomplete is enabled, hackers can directly steal your password from local storage. So autocomplete should not be enabled for the application.

CWE-525: Use of Web Browser Cache Containing Sensitive Information.

Tool Used to find: WebInspect

WebInspect Report Screenshot:

Privacy Violation: Autocomplete (11276)

Summary

Most recent browsers have features that will save password field content entered by users and then automatically complete password entry the next time the field are encountered. This feature is enabled by default and could leak password since it is stored on the hard drive of the user. The risk of this issue is greatly increased if users are accessing the application from a shared environment. Recommendations include setting autocomplete to "off" on all your password fields.

Please Note: Recent versions of most browsers, as noted below, now ignore the autocomplete="off" attribute for password fields in html forms. Users are allowed to decide the password policy at their own discretion using the password manager. Although setting is ineffective on these versions of browsers, it would continue to protect website users of earlier versions of these and other browsers that support this attribute.

Browsers NOT Supporting autocomplete="off":

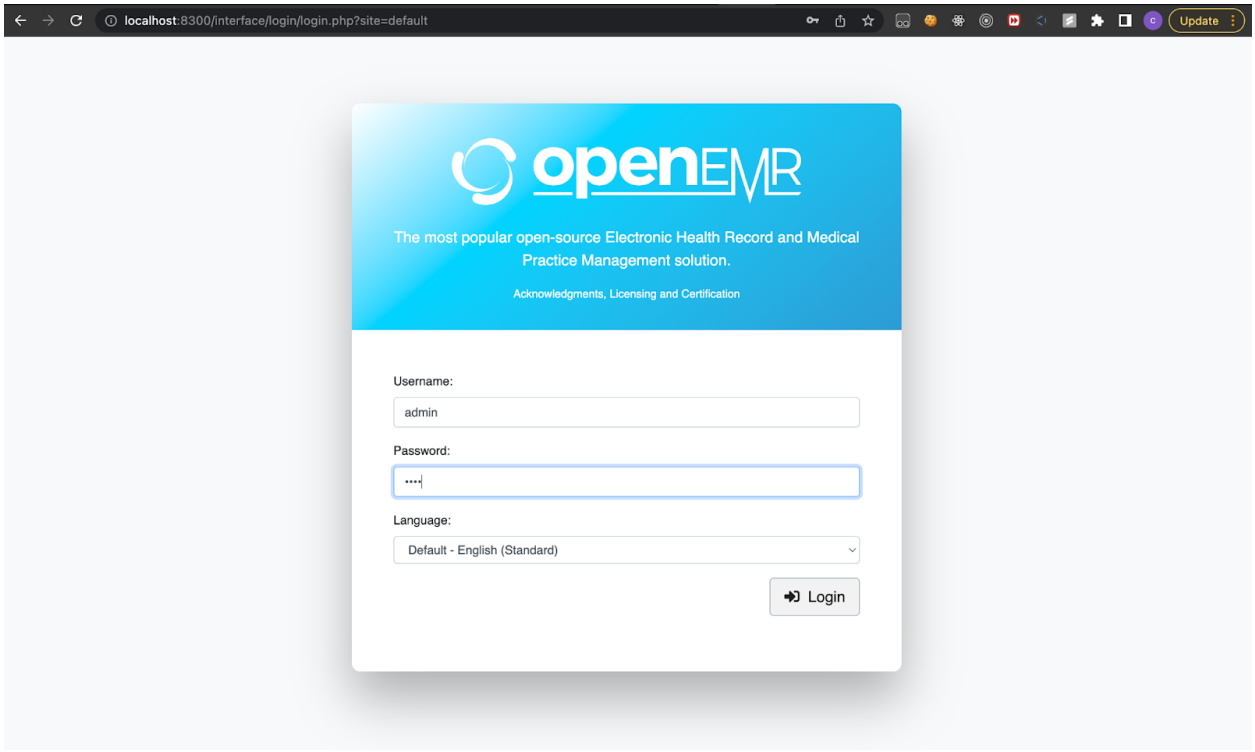
Internet Explorer version 11 or above
Firefox version 30 or above
Chrome version 34 or above
For other browsers, please refer to vendor specific documentation

Execution

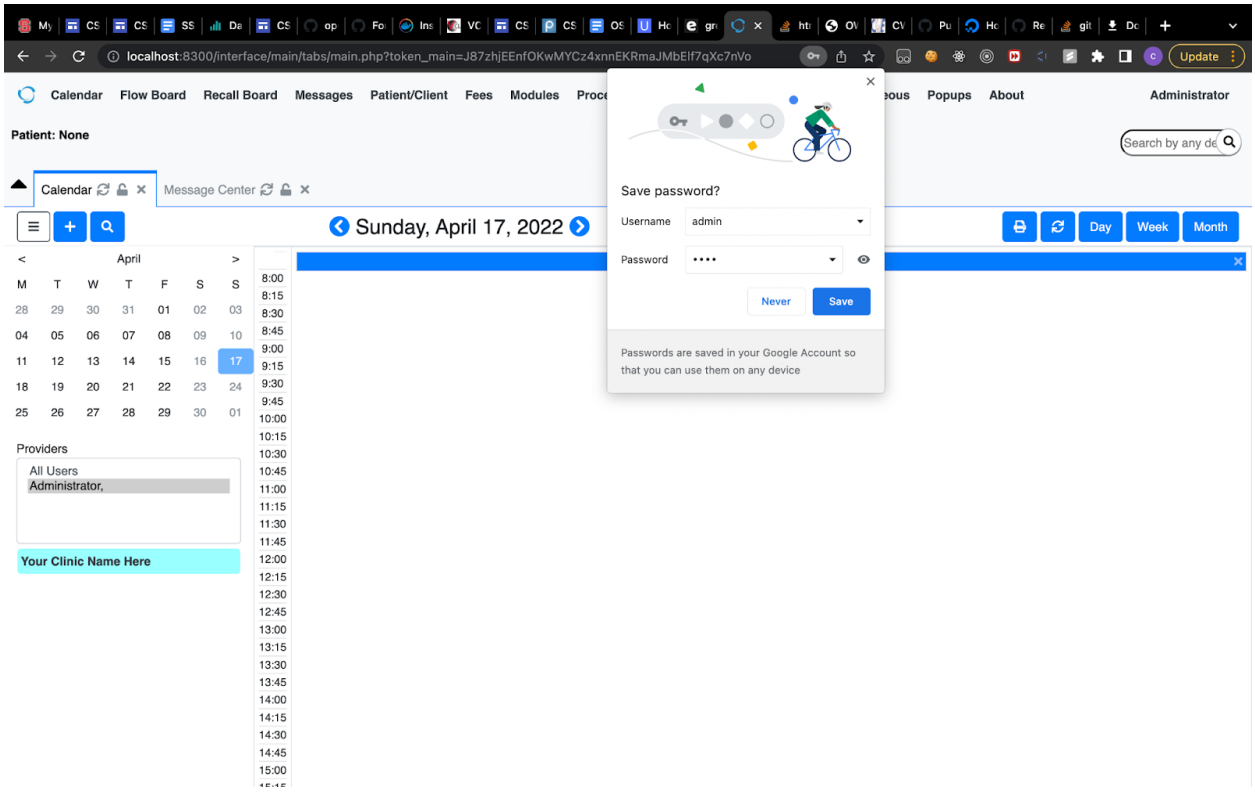
To verify if a password filed is vulnerable, first make sure to enable the autocomplete in your browser's settings, and then input the other fileds of the form to see whether the password is automatically filled. If yes, then it's vulnerable, otherwise, not. You may need to do it twice in case it is the first time you type in the credential in your browser.

Code Fix Suggestion: As all the modern browsers are well designed to detect password fields and store them in cache, there is practically no way to stop the browser from doing it. But we can trick the browser by making the password field as a text field whose appearance looks like a password but its type is still text and stop it from storing and autocompleting password fields.

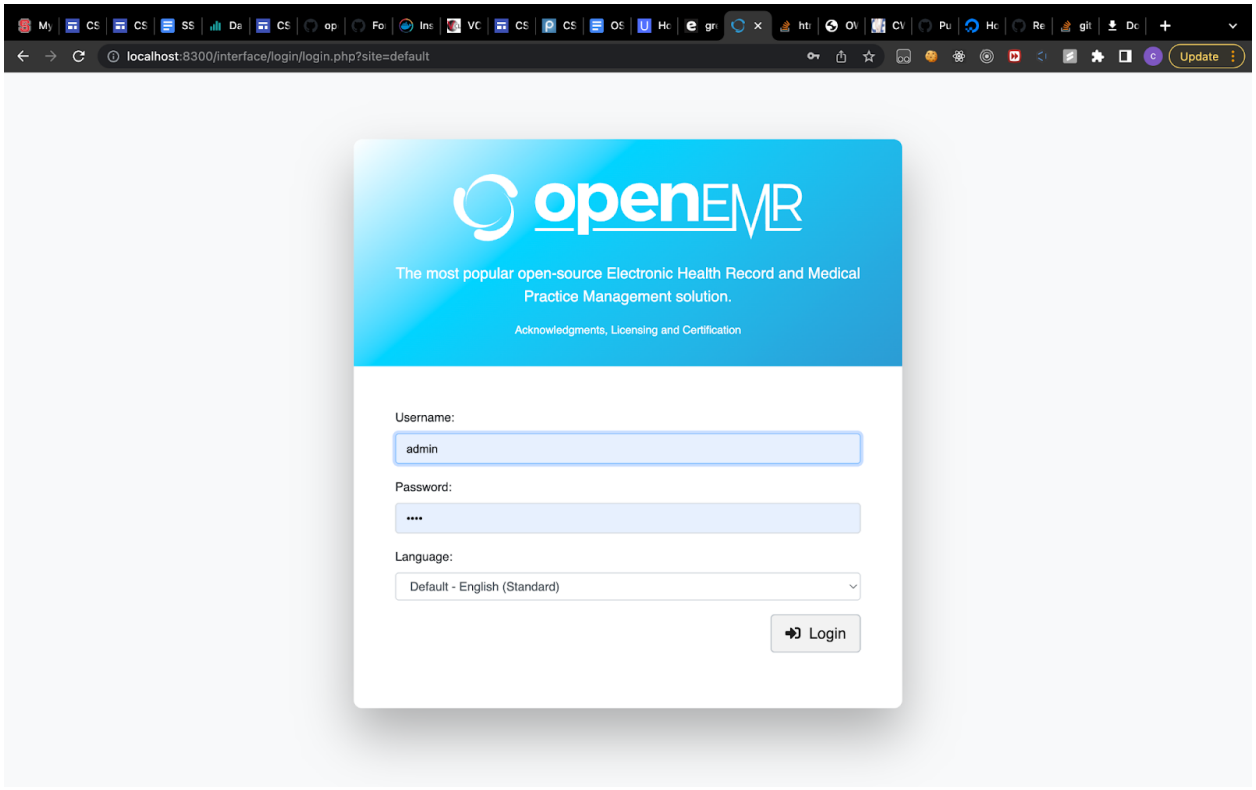
Before Fix:
Start openemr application in docker(username: admin, password: pass) or vcl(username: admin_openemr, password: admin_openemr) and login using corresponding credentials.



Once you successfully login you will see the browser asking to save password. Click on save and logout.



Now when you again land on to the browser login page it automatically fills the login details from the stored browser cache with any further typing.

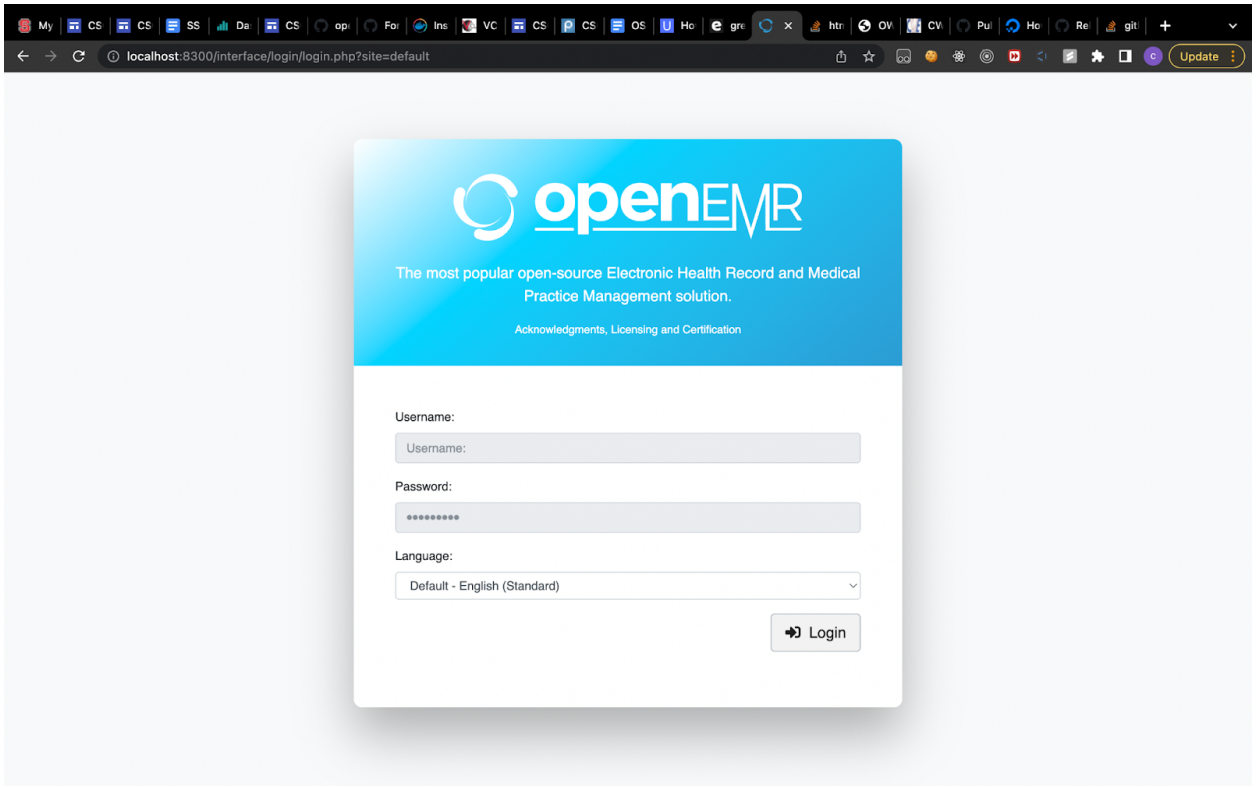


Existing Code in login.php responsible for this type of behavior:

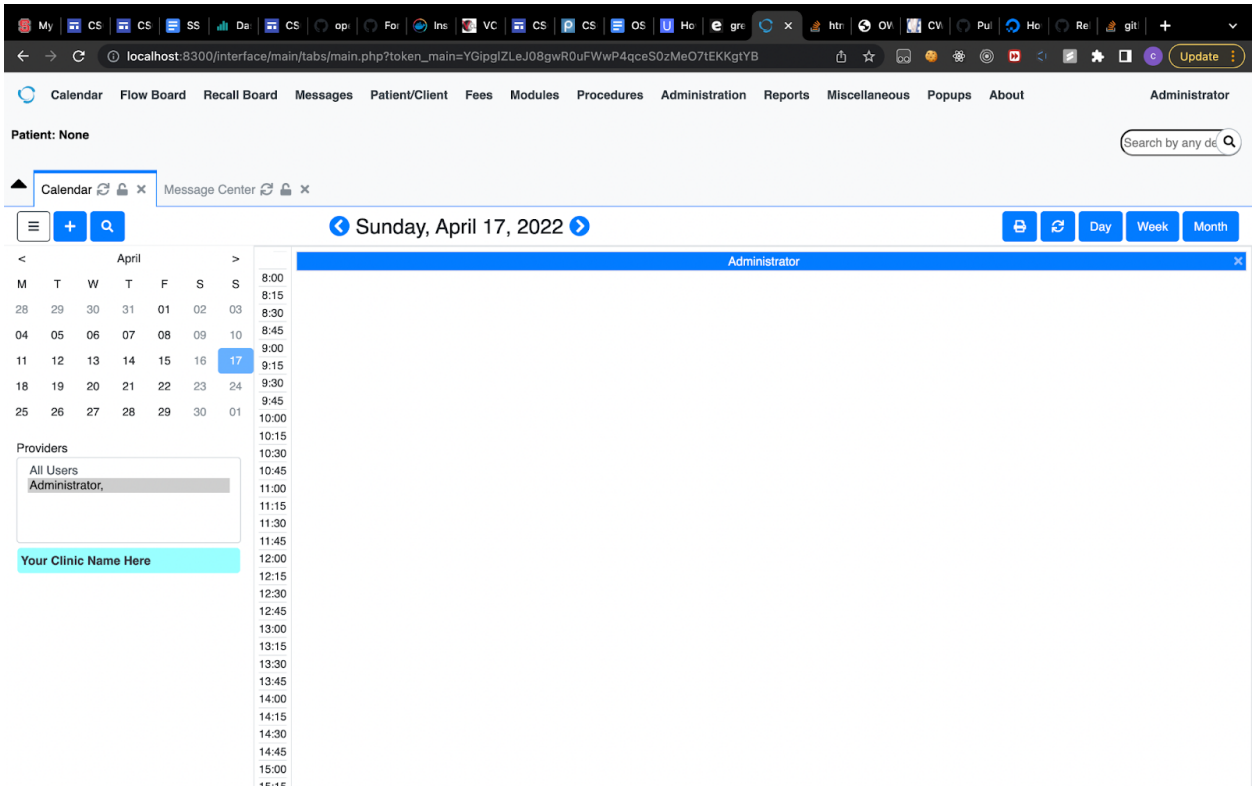
```
<?php } // End login failure block ?>
<div class="form-group">
    <label for="authUser" class="text-right"><?php echo xlt('Username:'); ?></label>
    <input type="text" class="form-control" id="authUser" name="authUser" placeholder="<?php echo
        xla('Username:'); ?>" />
</div>
<div class="form-group">
    <label for="clearPass" class="text-right"><?php echo xlt('Password:'); ?></label>
    <input type="password" class="form-control" id="clearPass" name="clearPass" placeholder="
    <?php echo xla('Password:'); ?>" />
</div>
```

Since the input type is password most of the modern browsers automatically try to store it in their local cache.

After fix:
Once you reload the openemr after the fix the UI appears like the below screenshot with default placeholder values in the username and password fields.



Here both the fields get active only when focused, preventing autocomplete. So the browser will not be able to autofill the credentials.



Once you successfully login using credentials mentioned in above section(**Before Fix** section) the browser will never ask to save passwords, as the browser was tricked by saying that the password field was of type text which exactly looks like a password to a human eye. Hence the browser will never store or autofill the login credentials in the login page.

Code changes in login.php for this fix:

This styling makes the text in the password field exactly like a password.

```
<style>
    @font-face {
      font-family: 'password';
      font-style: normal;
      font-weight: 400;
      src: url(https://jsbin-user-assets.s3.amazonaws.com/rafaelcastrocouto/password.ttf);
    }

    input#clearPass {
      font-family: 'password';
    }
</style>
```

This part of code turns off the autocomplete and makes fields read only when not focused.

```
<div class="form-group">
    <label for="authUser" class="text-right"><?php echo xlt('Username:'); ?></label>
    <input autocomplete="off" type="text" class="form-control" id="authUser" name="authUser"
    placeholder="<?php echo xla('Username:');?>" readonly onfocus="this.removeAttribute('readonly
        ');"/>
</div>
<div class="form-group">
    <label for="clearPass" class="text-right"><?php echo xlt('Password:'); ?></label>
    <input autocomplete="off" type="text" class="form-control" id="clearPass" name="clearPass"
    placeholder="<?php echo xla('Password:');?>" readonly onfocus="this.removeAttribute('readonly
        ');"/>
</div>
```

4. Weak Encryption: Insufficient Key Size

Reference: Project 1, Fortify True Positive 5.

Path to File: src\Common\Crypto\CryptoGen.php

Why it's a true positive: Here, the CoreDecrypt function is generating pre key and secret key which are of 32-bit length, The size of the resulting key is often insufficient and equivalent to the chance for a hash collision and brute force attacks.

```
213     private function coreDecrypt($sValue, $customPassword = null, $keySource = 'drive', $keyNumber = null)
214     {
215         $keyNumber = isset($keyNumber) ? $keyNumber : $this->keyVersion;
216
217         if (!extension_loaded('openssl')) {
218             error_log("OpenEMR Error : Decryption is not working because missing openssl extension.");
219             return false;
220         }
221
222         $raw = base64_decode($sValue, true);
223         if ($raw === false) {
224             error_log("OpenEMR Error : Encryption did not work because illegal characters were noted in base64_encoded data.");
225             return false;
226         }
227
228         if (empty($customPassword)) {
229             // Collect the encryption keys.
230             // The first key is for encryption. Then second key is for the HMAC hash
231             $sSecretKey = $this->collectCryptoKey($keyNumber, "a", $keySource);
232             $sSecretKeyHmac = $this->collectCryptoKey($keyNumber, "b", $keySource);
233         } else {
234             // customPassword mode, so turn the password keys
235             // The first key is for encryption. Then second key is for the HMAC hash
236             // First need to collect the salt from $raw (and then remove it from $raw)
237             $sSalt = mb_substr($raw, 0, 32, '8bit');
238             $raw = mb_substr($raw, 32, null, '8bit');
239             // Now turn the password into keys
240             $sPreKey = hash_pbkdf2('sha384', $customPassword, $sSalt, 100000, 32, true);
241             $sSecretKey = hash_hkdf('sha384', $sPreKey, 32, 'aes-256-encryption', $sSalt);
242             $sSecretKeyHmac = hash_hkdf('sha384', $sPreKey, 32, 'sha-384-authentication', $sSalt);
243         }
244
245         if (empty($sSecretKey) || empty($sSecretKeyHmac)) {
246             error_log("OpenEMR Error : Encryption is not working because key(s) is blank.");
247             return false;
248         }
249
250         $ivLength = openssl_cipher_iv_length('aes-256-cbc');
251         $hmacHash = mb_substr($raw, 0, 48, '8bit');
252         $iv = mb_substr($raw, 48, $ivLength, '8bit');
253         $encrypted_data = mb_substr($raw, ($ivLength + 48), null, '8bit');
254
255         $calculatedHmacHash = hash_hmac('sha384', $iv . $encrypted_data, $sSecretKeyHmac, true);
256
257         if (hash_equals($hmacHash, $calculatedHmacHash)) {
258             return openssl_decrypt(
259                 $encrypted_data,
```

Code Sample (Fortify - Project 1)

Fix for the Vulnerability: The standard recommends a salt length of at least 64 bits The US National Institute of Standards and Technology recommends a salt length of 128 bits or higher

CWE: CWE-326: Inadequate Encryption Strength

ASVS Control: Level 2,3

Test Case id: 6.2.4

Description: Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers, or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks

Code Sample

Repeatable Steps:

1. Login to vcl and start the openemr application.
2. Login as admin. (username: admin_openemr password: admin_openemr)
3. Select Administrator > Change Password
4. Enter the current password
5. Enter a new password that complies with the OpenEMR guidelines (e.g. aA!123456)
6. Logout
7. Log back in as admin using the new password
8. Make the code fixes below
9. Repeat steps 2-7

Expected Results: The new salt length does not change any login/logout functionality

Code Fix: For this vulnerability, the length of the PreKey was changed from 32 bits to the NIST recommendation of 128 bits. The references to PreKey in SecretKey and SecretKeyHmac were also modified to account for the larger PreKey. This change was made in both the coreEncrypt and coreDecrypt functions in CryptoGen.php.

```
33  33  class CryptoGen
34  34  {
35  35      # This is the current encrypt/decrypt version
36  36      # (this will always be a three digit number that we will
37  37      #  increment when update the encrypt/decrypt methodology
38  38      #  which allows being able to maintain backward compatibility
39  39      #  to decrypt values from prior versions)
40  40      # Remember to update cryptCheckStandard() and decryptStandard()
41  41      #  when increment this.
42  42      private $encryptionVersion = "006";
43  43      # This is the current key version. As above, will increment this
44  44      #  when update the encrypt/decrypt methodology to allow backward
45  45      #  compatibility.
46  46      # Remember to update decryptStandard() when increment this.
47  47      private $keyVersion = "six";
48  48
    49+      # Length in bits of the PreKey in coreEncrypt() and coreDecrypt()
    50+      private $preKeyLength = 128;
    51+
49  52      # Note that dynamic variables in this class in the collectCryptoKey()
50  53      #  function are used to store the key cache.
```

Modification of PreKey Size()

```
142 145     private function coreEncrypt($sValue, $customPassword = null, $keySource = 'drive', $keyNumber = null)
143 146     {
144 147         $keyNumber = isset($keyNumber) ? $keyNumber : $this->keyVersion;
145 148
146 149         if (!extension_loaded('openssl')) {
147 150             error_log("OpenEMR Error : Encryption is not working because missing openssl extension.");
148 151         }
149 152
150 153         if (empty($customPassword)) {
151 154             // Collect the encryption keys. If they do not exist, then create them
152 155             // The first key is for encryption. Then second key is for the HMAC hash
153 156             $ssSecretKey = $this->collectCryptoKey($keyNumber, "a", $keySource);
154 157             $ssSecretKeyHmac = $this->collectCryptoKey($keyNumber, "b", $keySource);
155 158         } else {
156 159             // customPassword mode, so turn the password into keys
157 160             $ssSalt = RandomGenUtils::produceRandomBytes(32);
158 161             if (empty($ssSalt)) {
159 162                 error_log('OpenEMR Error : Random Bytes error - exiting');
160 163                 die();
161 164             }
162     -           $sPreKey = hash_pbkdf2('sha384', $customPassword, $ssSalt, 100000, 32, true);
163     -           $ssSecretKey = hash_hkdf('sha384', $sPreKey, 32, 'aes-256-encryption', $ssSalt);
164     -           $ssSecretKeyHmac = hash_hkdf('sha384', $sPreKey, 32, 'sha-384-authentication', $ssSalt);
165+             $sPreKey = hash_pbkdf2('sha384', $customPassword, $ssSalt, 100000, $preKeyLength, true);
166+             $ssSecretKey = hash_hkdf('sha384', $sPreKey, $preKeyLength, 'aes-256-encryption', $ssSalt);
167+             $ssSecretKeyHmac = hash_hkdf('sha384', $sPreKey, $preKeyLength, 'sha-384-authentication', $ssSalt);
165 168         }
```

Modification of coreEncrypt()

```
213 216     private function coreDecrypt($sValue, $customPassword = null, $keySource = 'drive', $keyNumber = null)
214 217     {
215 218         $keyNumber = isset($keyNumber) ? $keyNumber : $this->keyVersion;
216 219
217 220         if (!extension_loaded('openssl')) {
218 221             error_log("OpenEMR Error : Decryption is not working because missing openssl extension.");
219 222             return false;
220 223         }
221 224
222 225         $raw = base64_decode($sValue, true);
223 226         if ($raw === false) {
224 227             error_log("OpenEMR Error : Encryption did not work because illegal characters were noted in base64_encoded data.");
225 228             return false;
226 229         }
227 230
228 231         if (empty($customPassword)) {
229 232             // Collect the encryption keys.
230 233             // The first key is for encryption. Then second key is for the HMAC hash
231 234             $ssSecretKey = $this->collectCryptoKey($keyNumber, "a", $keySource);
232 235             $ssSecretKeyHmac = $this->collectCryptoKey($keyNumber, "b", $keySource);
233 236         } else {
234 237             // customPassword mode, so turn the password keys
235 238             // The first key is for encryption. Then second key is for the HMAC hash
236 239             // First need to collect the salt from $raw (and then remove it from $raw)
237 240             $ssSalt = mb_substr($raw, 0, 32, '8bit');
238 241             $raw = mb_substr($raw, 32, null, '8bit');
239 242             // Now turn the password into keys
240     -           $sPreKey = hash_pbkdf2('sha384', $customPassword, $ssSalt, 100000, 32, true);
241     -           $ssSecretKey = hash_hkdf('sha384', $sPreKey, 32, 'aes-256-encryption', $ssSalt);
242     -           $ssSecretKeyHmac = hash_hkdf('sha384', $sPreKey, 32, 'sha-384-authentication', $ssSalt);
243+             $sPreKey = hash_pbkdf2('sha384', $customPassword, $ssSalt, 100000, $preKeyLength, true);
244+             $ssSecretKey = hash_hkdf('sha384', $sPreKey, $preKeyLength, 'aes-256-encryption', $ssSalt);
245+             $ssSecretKeyHmac = hash_hkdf('sha384', $sPreKey, $preKeyLength, 'sha-384-authentication', $ssSalt);
```

Modification of coreDecrypt()

4. Penetration Testing

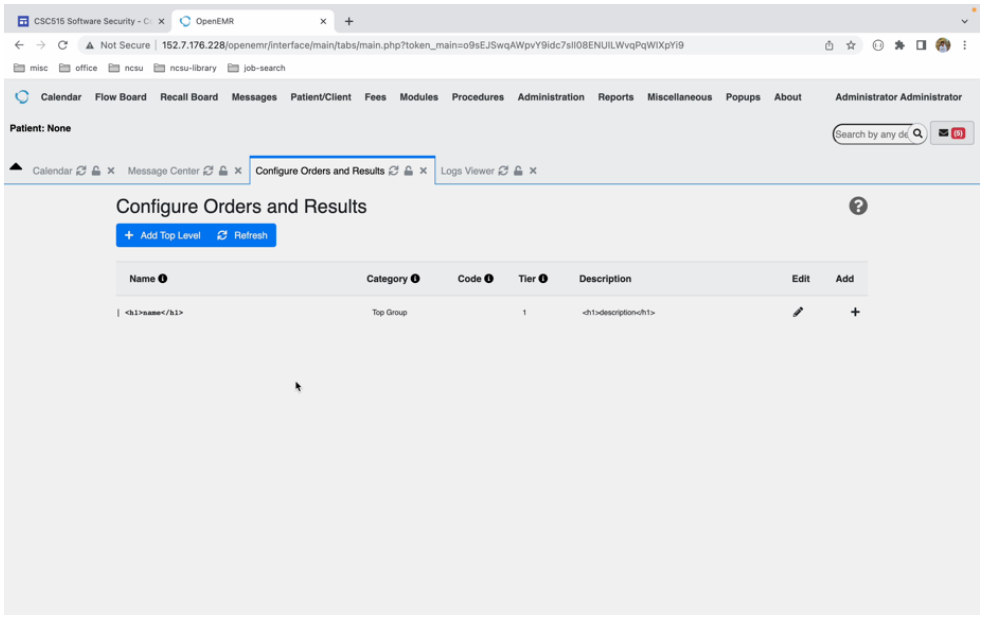
1. Test Case ID: 5.3.5

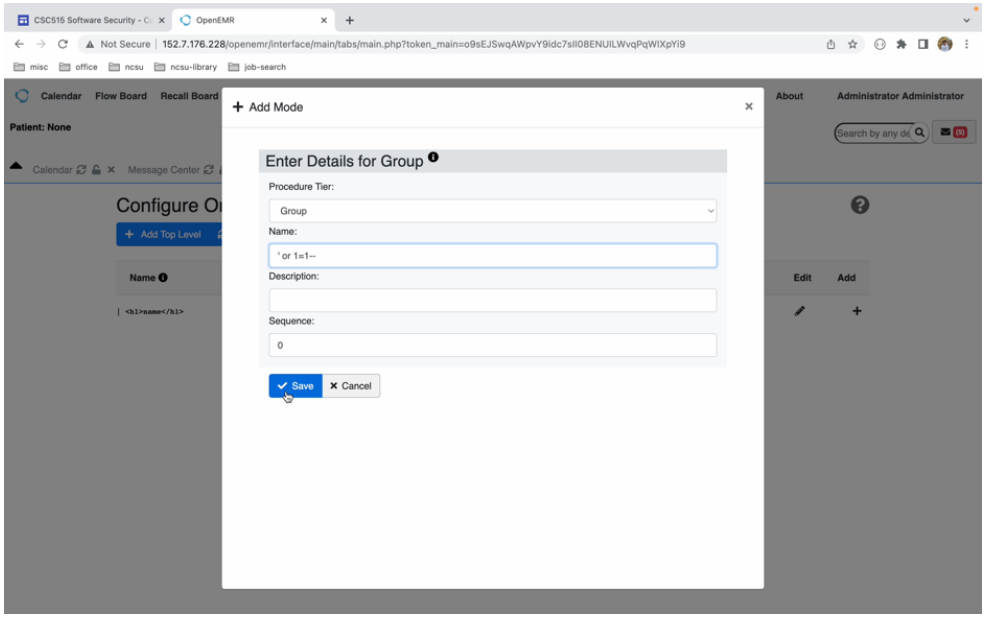
Description: Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.

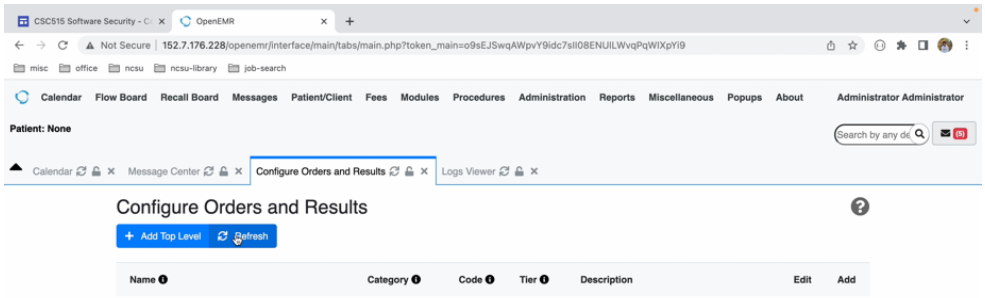
Repeatable Step:
1. Launch the openemr application in vcl.
2. Open http://localhost/openemr in Firefox browser within vcl.
3. Login as admin. (username: admin_openemr Password: admin_openemr)
4. On the top header, hover over Procedures and click on Configuration.
5. Click on “Add Top Level” button.
6. Select Procedure Tier: Group, Name: hello, Description: description, Sequence: 0
7. Click on Save.
8. Again click on “Add Top Level” button.
9. Now select Procedure Tier: Group, Name: **‘ OR 1=1--,** Description: description, Sequence: 0
10. Click on Save.
11. Click on the Refresh button.
12. Notice the change.

Expected Results:
1. The new group with name should not be saved and it should not affect other groups.

CWE Info:
1. 89







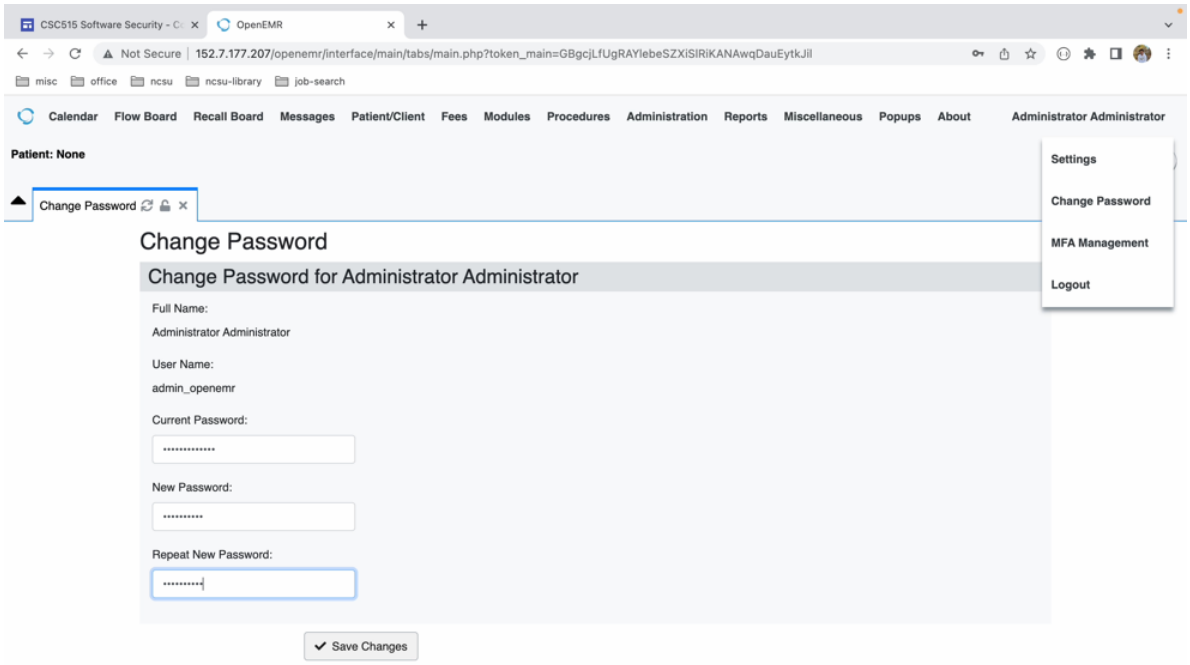
2. Test Case ID: 2.10.4

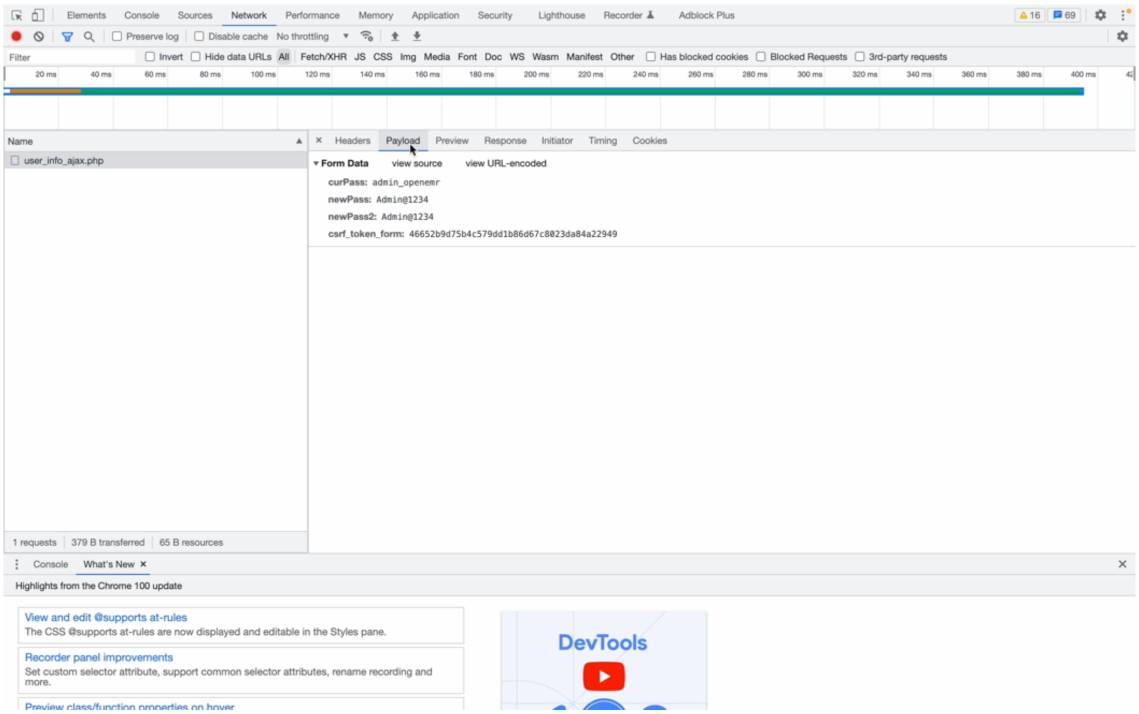
Description: Verify passwords, integrations with databases and third- party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store (L1), hardware TPM, or an HSM (L3) is recommended for password storage.

Repeatable Step:
1. Launch the openemr application in vcl.
2. Open http://localhost/openemr in Firefox browser within vcl.
3. Login as admin. (username: admin_openemr Password: admin_openemr)
4. On the top header, hover over Administrator Administrator and click on change password.
5. Right click on the screen and click on inspect. Monitor the network tab.
6. Type current password as admin_openemr.
7. Type new password as Admin@1234 and repeat new password as Admin@1234.
8. Open the network tab and select the http://localhost/openemr/interface/usergroup/user_info_ajax.php URL.
9. Click on Payload and look at curPass, newPass, newPass2 in the payload.

Expected Results: The current password, new password, repeat new password should be masked in the network tab.

CWE Info: 798





Vulnerability #	Elapsed Time	Ref Info for Video Traceability	CWE	Commentary
1	About 3 minutes	1:46:40	89	SQL Injection
2	About 1 minute	2:55:42	798	Password vulnerability

3. Test Case ID: 5.4.1-1

Description: Verify that the application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.

Repeatable Step:

1. Login as any user (Eg: username: admin password: pass)
2. In the top navigation bar, hover over the administration tab then hover over the clinic tab
3. In that select Facilities and click on it.
4. Click on Add facilities Button.
5. Under the name of the facilities enter a input that a lot of characters for eg.,

(“*aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa*

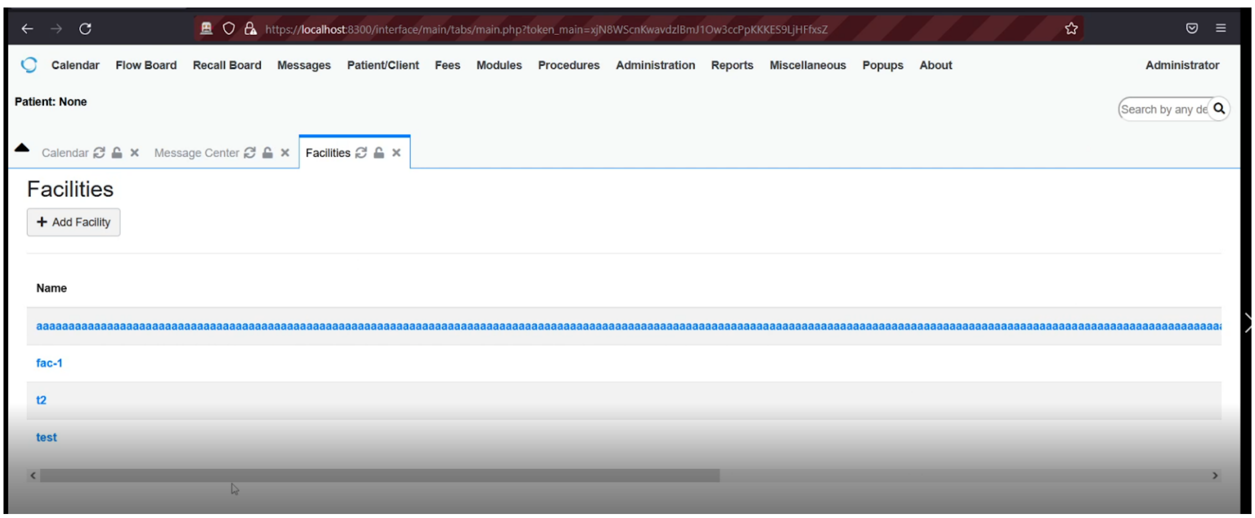
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”)

6. Enter other mandatory information in the form like color and click on save button.
7. Now the entered facility details will be listed as shown in the below screenshot.

Expected Results:

- The given request should be rejected or possibly truncated if incase the request is accepted.

CWE Info: 120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')



V7.4 Error Handling

---

4. Test Case ID: 7.4.1-1

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.

Repeatable Step:

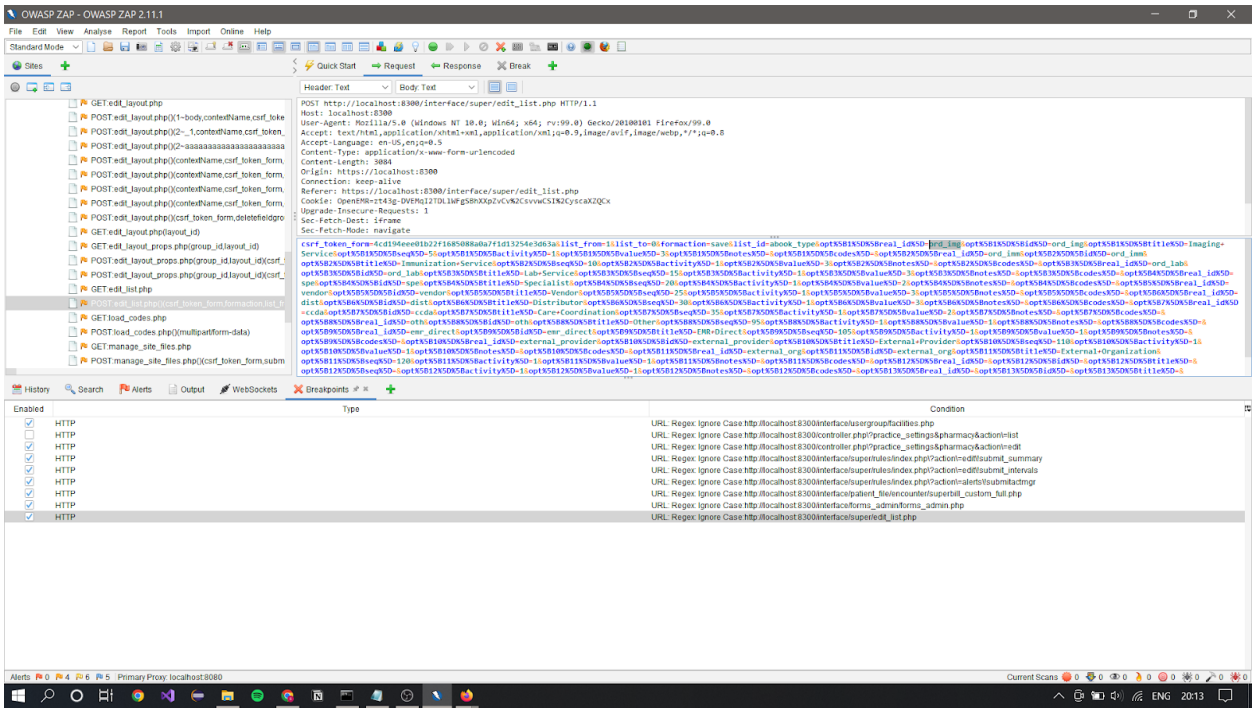
1. Login as any user (Eg: username: admin   password: pass)
2. Under the **Administration** tab, hover the mouse over **Form.**
3. Click on the Lists tab to list all the details.

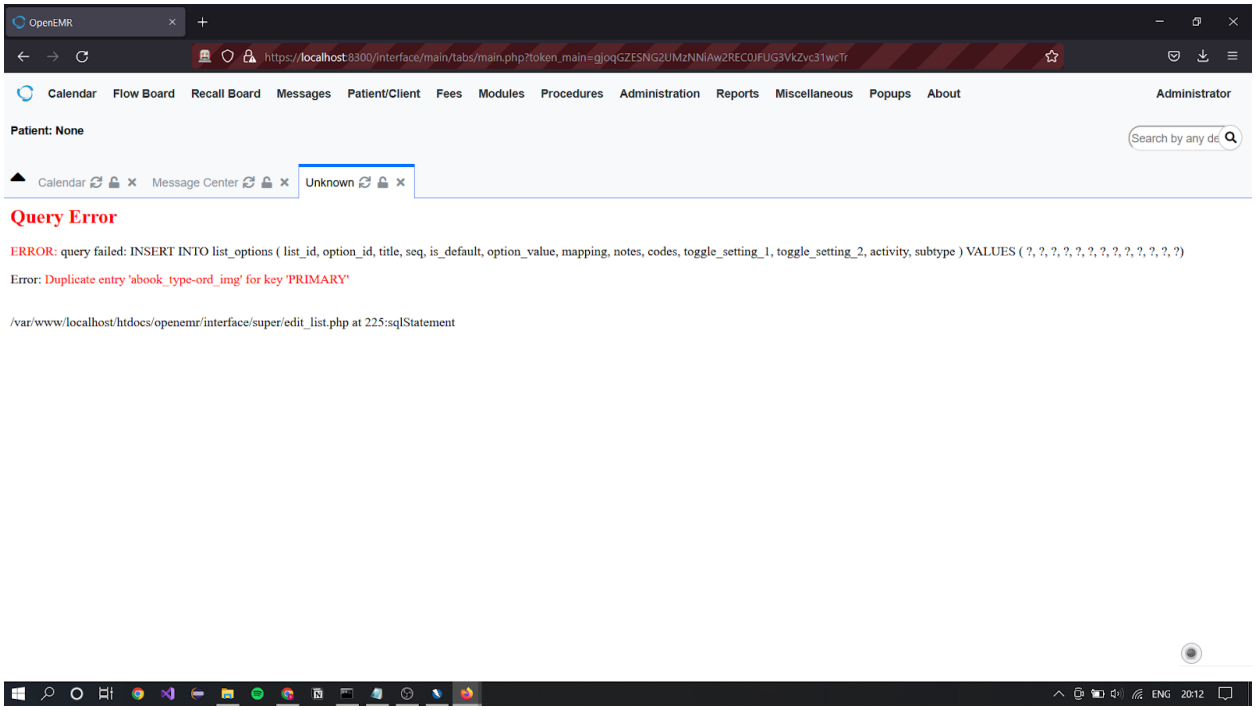
4. Click on save to retrieve the API endpoint.
5. Now, find the respective endpoint being called which is **http://<ip-address>:8300/interface/super/edit_list.php**
6. Now, place a breakpoint on the above mentioned endpoint using ZAP as a proxy.
7. Repeat steps 3-7
8. After Clicking save, come over to zap and change the attributes (opt[1][real_id]) from the value **ord_img** to **1235**
9. Press on the play button in zap for sending the API request.
10. We can see the error being displayed in the UI showing all the sensitive information about the table and its respective columns.

Expected Results:

- The system should show a predefined/ generic error message which doesn't expose any sensitive information.

CWE Info: 209: Self-generated Error Message Containing Sensitive Information





V7.1 Log Content

---

5. Test Case ID: 12.2.1-1

Description: Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.

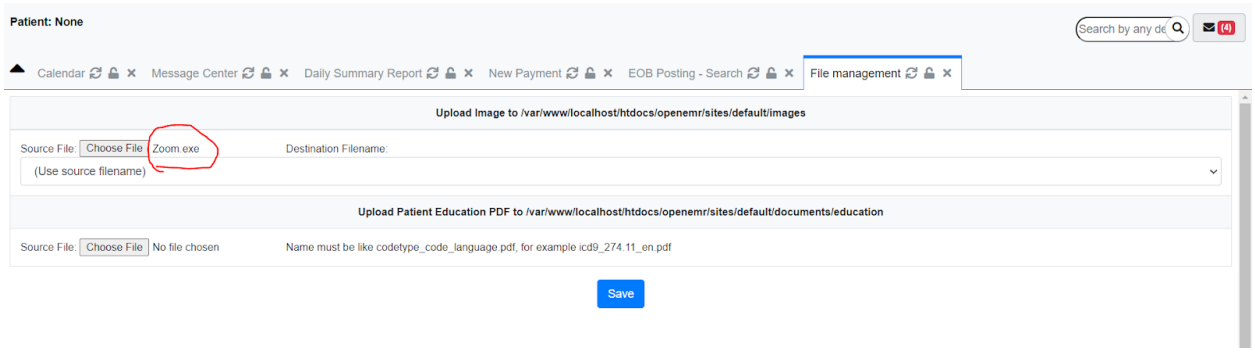
Repeatable Step:

1. Login as any user (Eg: username: admin password: pass)
2. Under the **Administration** tab, hover the mouse over **System.**
3. Click on **Files** tab, we can see two upload forms one for images and another for pdf's.
4. Under **Upload Image to /var/www/localhost/htdocs/openemr/sites/default/images,** Choose a file type such as exe, .bat other than image types like jpeg, jpg, png.
5. Click on save button below to make an upload API request to process the file.
6. Also, Under **Upload Patient Education PDF to /var/www/localhost/htdocs/openemr/sites/default/documents/education** repeat the steps 4 and 5 to upload file formats that are not PDF

Expected Results:

- The system should restrict the uploading of dangerous file types with front end validations incorporated.

CWE Info: 434: Unrestricted Upload of File with Dangerous Type



V1.5 Input and Output Architecture

---

6. Test Case ID: 1.5.3-1

Description: Verify that input validation is enforced on a trusted service layer.

Repeatable Step:

1. Login as any user (Eg: username: admin password: pass)
2. Under the **Administration** tab, hover the mouse over **Practise.**
3. Click on **Rules** tab, In that click on **Add new** button
4. Enter the **title** field as rule_1.
5. Under this field select the radio button **Patient Reminder** and then click on save.
6. Under Rule Interval enter all the input elements (For example: 1) with the time unit to be the default value month.
7. Click on Save.
8. Now, find the respective endpoint being called which is

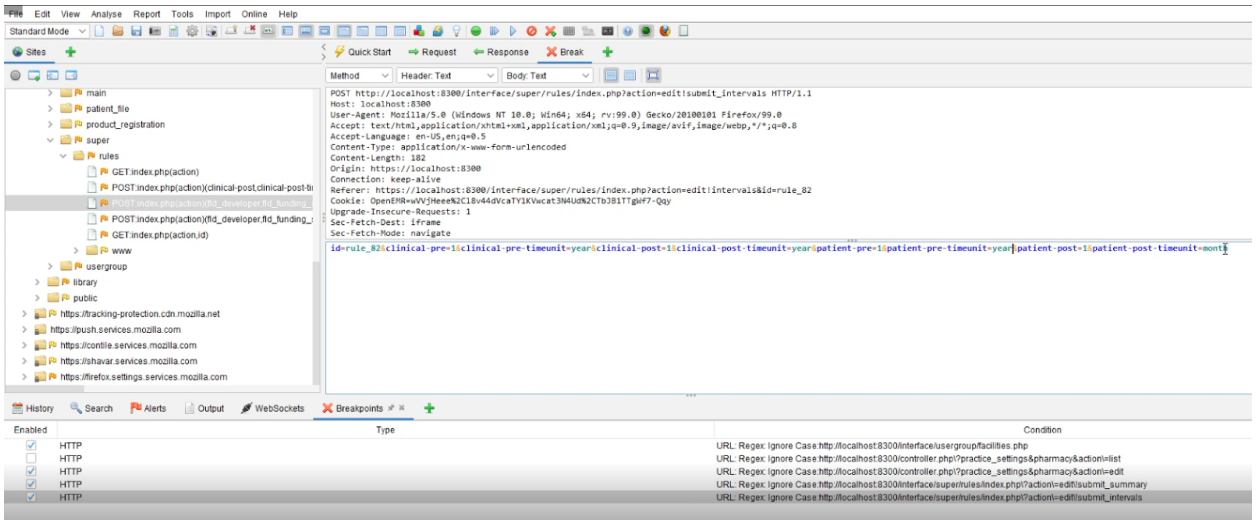
http://<ip-address>:8300/interface/super/rules/index.php?action=edit!submit_intervals

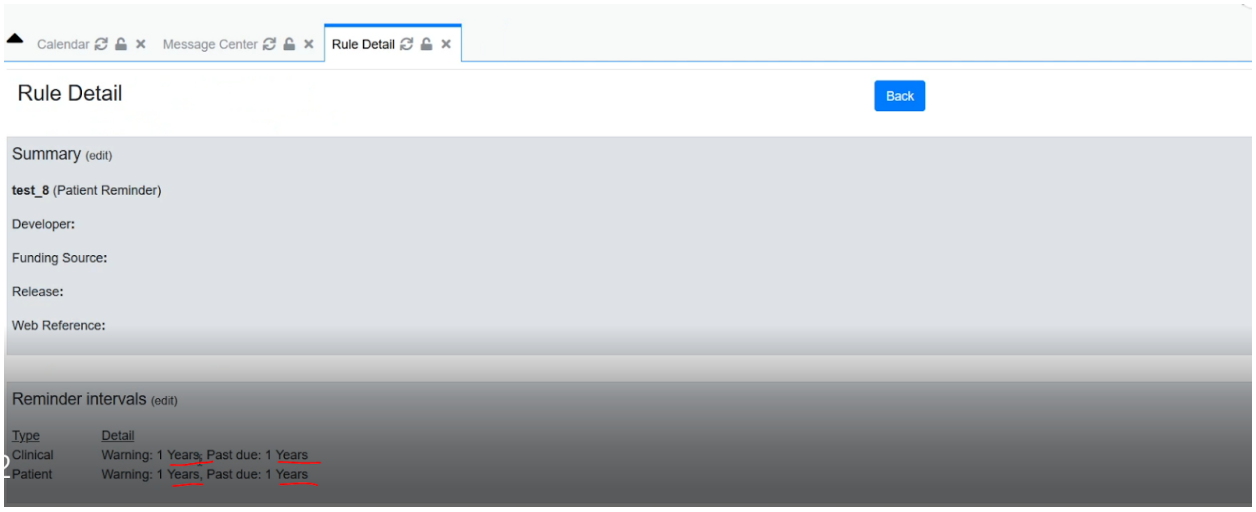
9. Now, place a breakpoint on the above mentioned endpoint using ZAP as a proxy.
10. Repeat steps 3-7
11. After Clicking save, come over to zap and change the attributes (Clinical-pre-time, clinical-post-time, patient-post-time) as **year**
12. Press on the play button in zap for sending the API request.
13. The Rule gets successfully added and shown on the listing page.

Expected Results:

- The year option should not be accepted as it is not a valid input.

CWE Info: 602: Client-Side Enforcement of Server-Side Security





V1.5 Input and Output Architecture

---

7. Test Case ID: 1.5.3-2

Description: Verify that input validation is enforced on a trusted service layer.

Repeatable Step:

1. Login as any user (Eg: username: admin password: pass)
2. Under the **Administration** tab, hover the mouse over **System.**
3. Click on **Language** tab, In that click on **Add new Language** tab
4. Enter the value ‘EN’ in the input element language code and
5. Under the input element Language Name, enter the value ‘English’.
6. Click on **Add**.
7. Now, find the respective endpoint being called which is

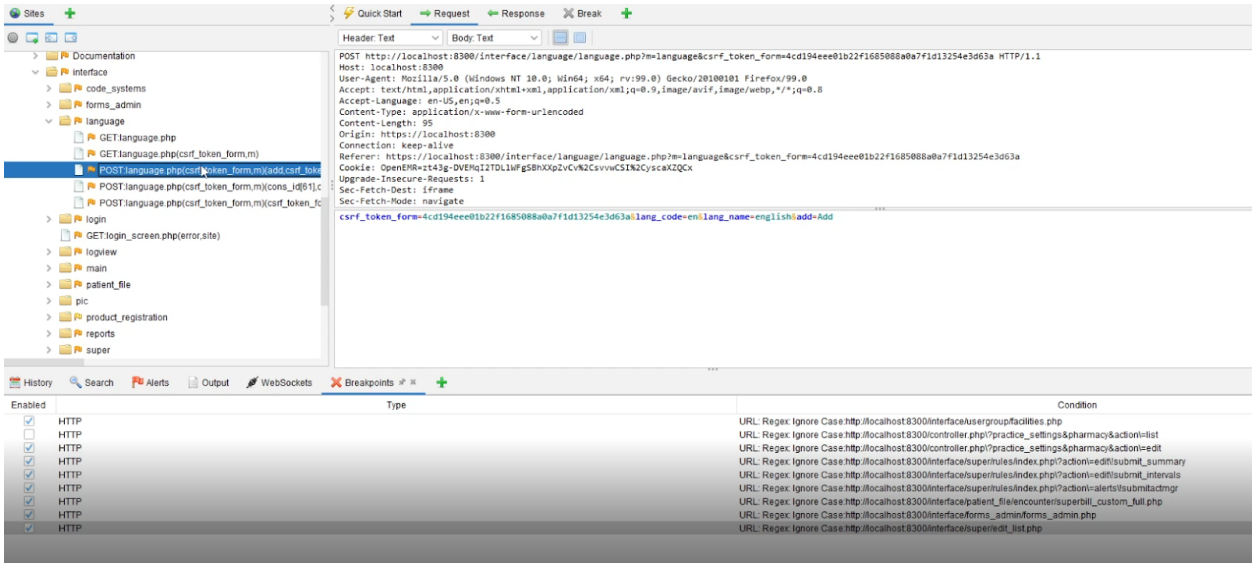
http://<ip-address>:8300/interface/language/language.php?m=language&csrf_token_form=1283d3a99ad5614b22e9d4f666bd9a7f3ac7c146

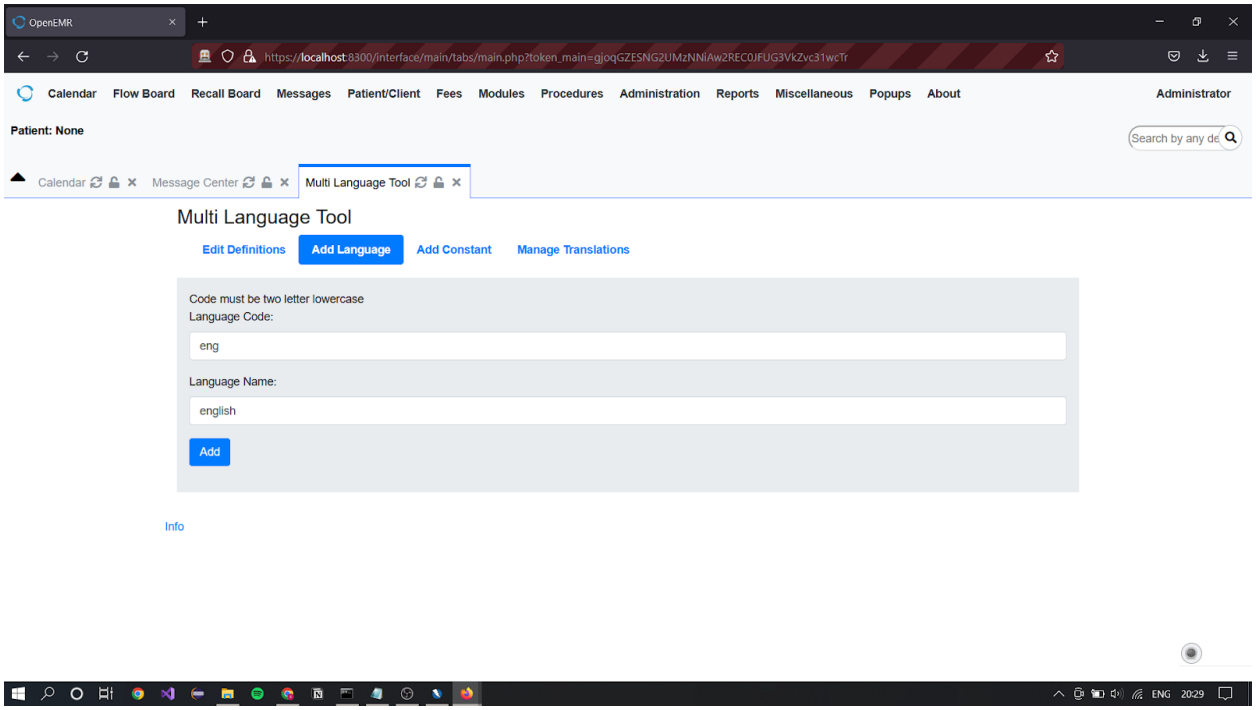
8. Now, place a breakpoint on the above mentioned endpoint using ZAP as a proxy.
9. Repeat steps 3-6
10. After Clicking **Add**, come over to zap and change the attributes (**lang_code**) as other values like **‘eng’** which is actually bypassing the client validations and also other potentially harmful XSS input like **<img src=javascript:alert(“XSS”)>**
11. Press on the play button in zap for sending the API request.
12. The language gets successfully added with the above given inputs and is listed after saving.

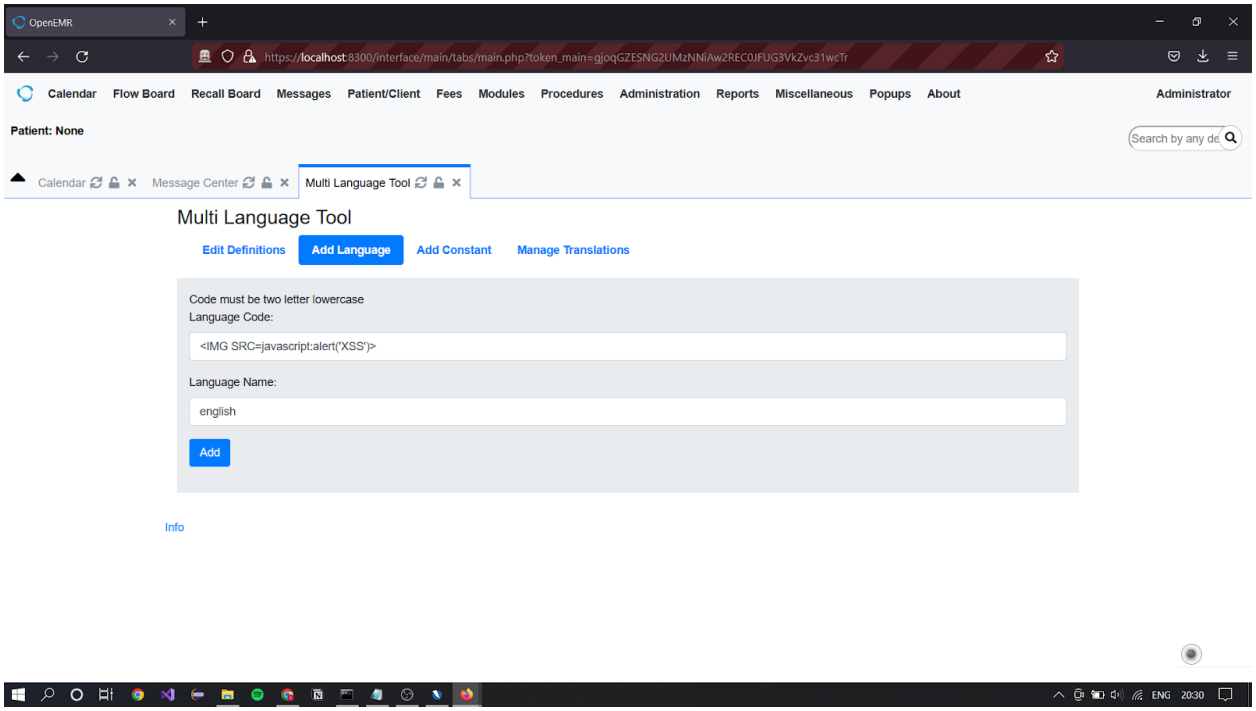
Expected Results:

- The new language creation API request should fail with relevant status code because the language code attribute is not a valid value.

CWE Info: 602: Client-Side Enforcement of Server-Side Security
79: Improper Neutralization of Input During Web Page Generation




OpenEMR
https://localhost:8300/interface/main/tabs/main.php?token_main=gjoqGZESNG2UMzNNiAw2REC0JFUG3VkZvc31wcTr
Calendar Flow Board Recall Board Messages Patient/Client Fees Modules Procedures Administration Reports Miscellaneous Popups About Administrator
Patient: None
Search by any de
Calendar Message Center Multi Language Tool
Multi Language Tool
Edit Definitions Add Language Add Constant Manage Translations
Code must be two letter lowercase
Language Code:
eng
Language Name:
english
Add
Info
ENG 20:29


OpenEMR
https://localhost:8300/interface/main/tabs/main.php?token_main=gjoqGZESNG2UMzNNiAw2REC0JFUG3VkZvc31wcTr
Calendar Flow Board Recall Board Messages Patient/Client Fees Modules Procedures Administration Reports Miscellaneous Popups About Administrator
Patient: None
Search by any de
Calendar Message Center Multi Language Tool
Multi Language Tool
Edit Definitions Add Language Add Constant Manage Translations
Code must be two letter lowercase
Language Code:
<IMG SRC=javascript:alert('XSS')>
Language Name:
english
Add
Info
ENG 20:30

# of Vulnerability	Elapsed Time	Ref Info for Video Traceability	CWE	Commentary
3	46th Min	Vid #2 - 21th Min 24 Sec	120	
4	About 2 hours 10 Mins	Vid #3 - 5 Min 51 Sec	209	
5	About 2 hours 18th Min	Vid #3 - 14 Min 09 Sec	434	
6	About 1 Hour 12 Min	Vid #2 - 47th Min	602	
7	About 2 hours 27th Min	Vid #3 - 23 Min	79, 602	

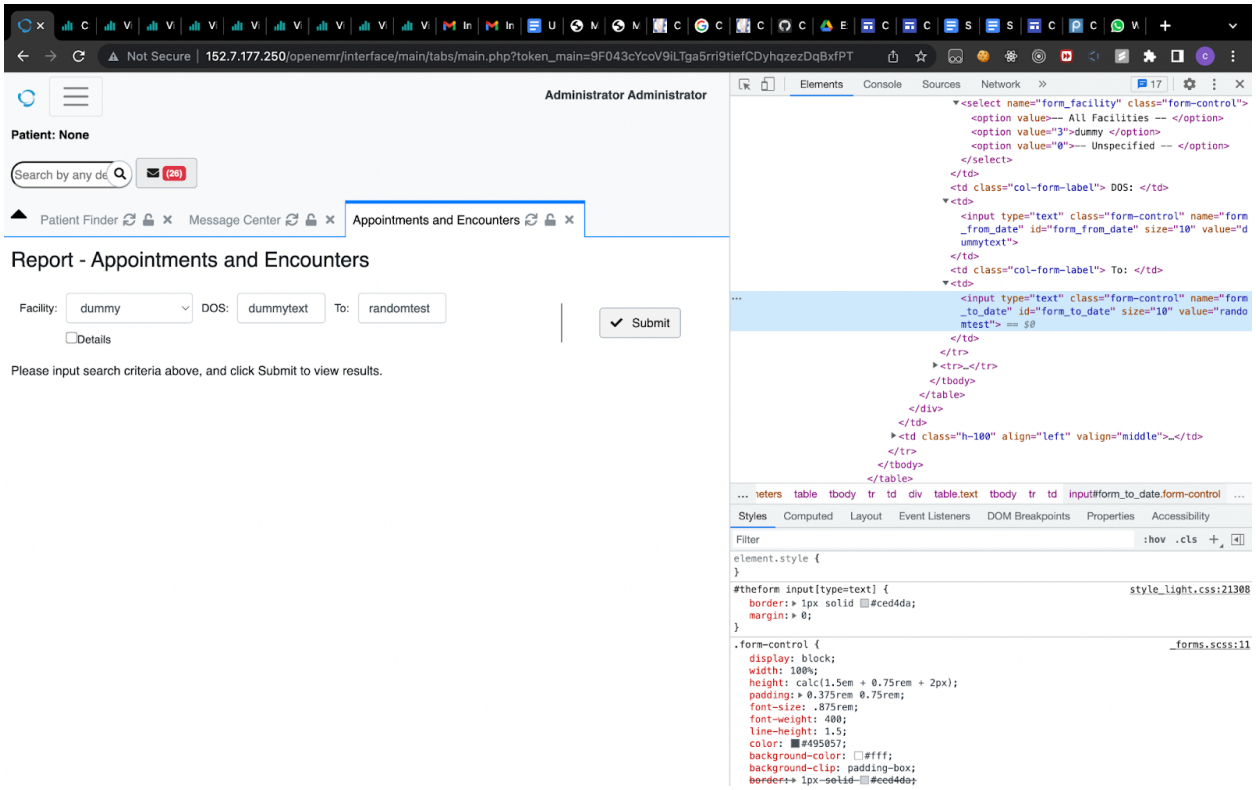
V1.5 Input and Output Architecture

---

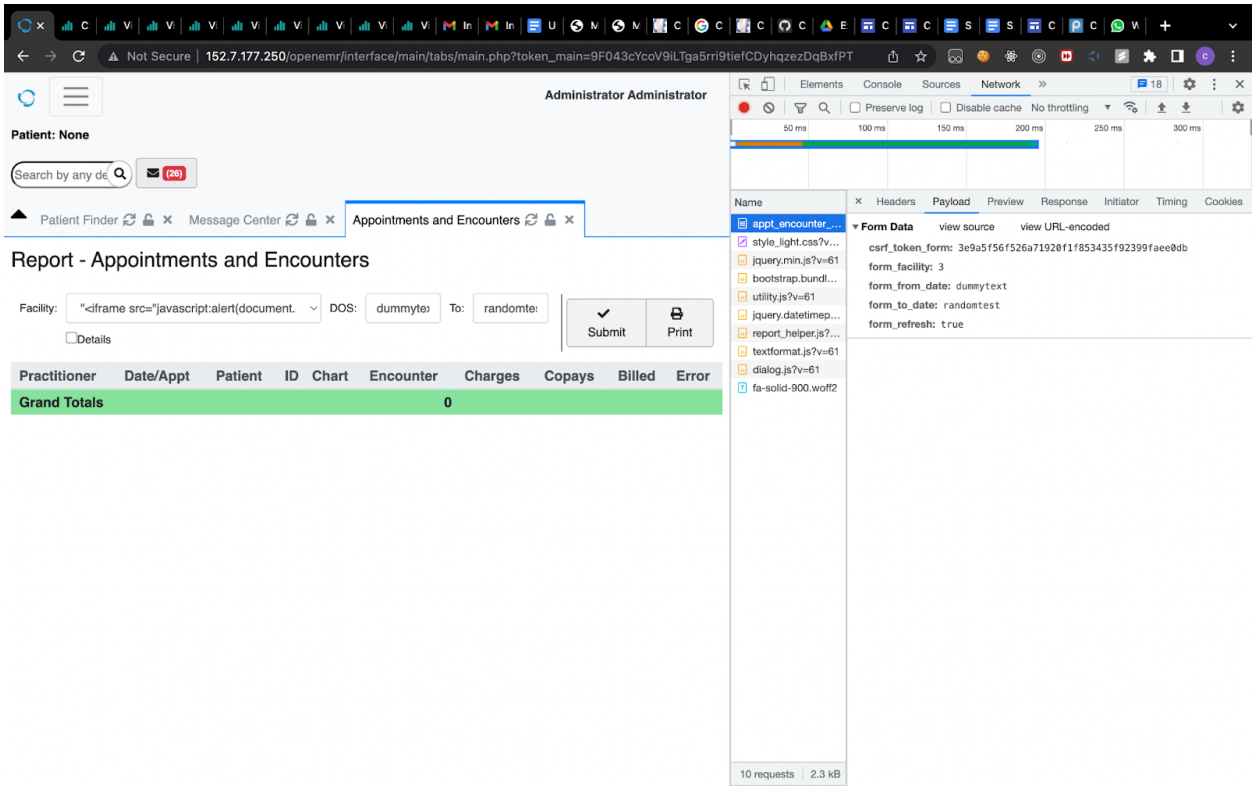
8. Test Case ID: 1.5.1

Description: Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance

Repeatable Steps:
1. Login as any user (Eg: username: admin password: pass)
2. In the top navigation bar, hover over the Reports tab then hover over the Visits tab and select Appt-Enc.
3. Now you will see 3 fields on the page (Facility, DOS: and To:) and inspect each of them one by one and make changes in the browser html as shown in the below screenshot.



4. Once the changes are done navigate to the network tab and clear the network calls already stacked.
5. Once the network tab is empty click on the Submit button.
6. Now you should be able to see a POST call being made to the backend with the tampered input values which we have given as shown in the below screenshot.



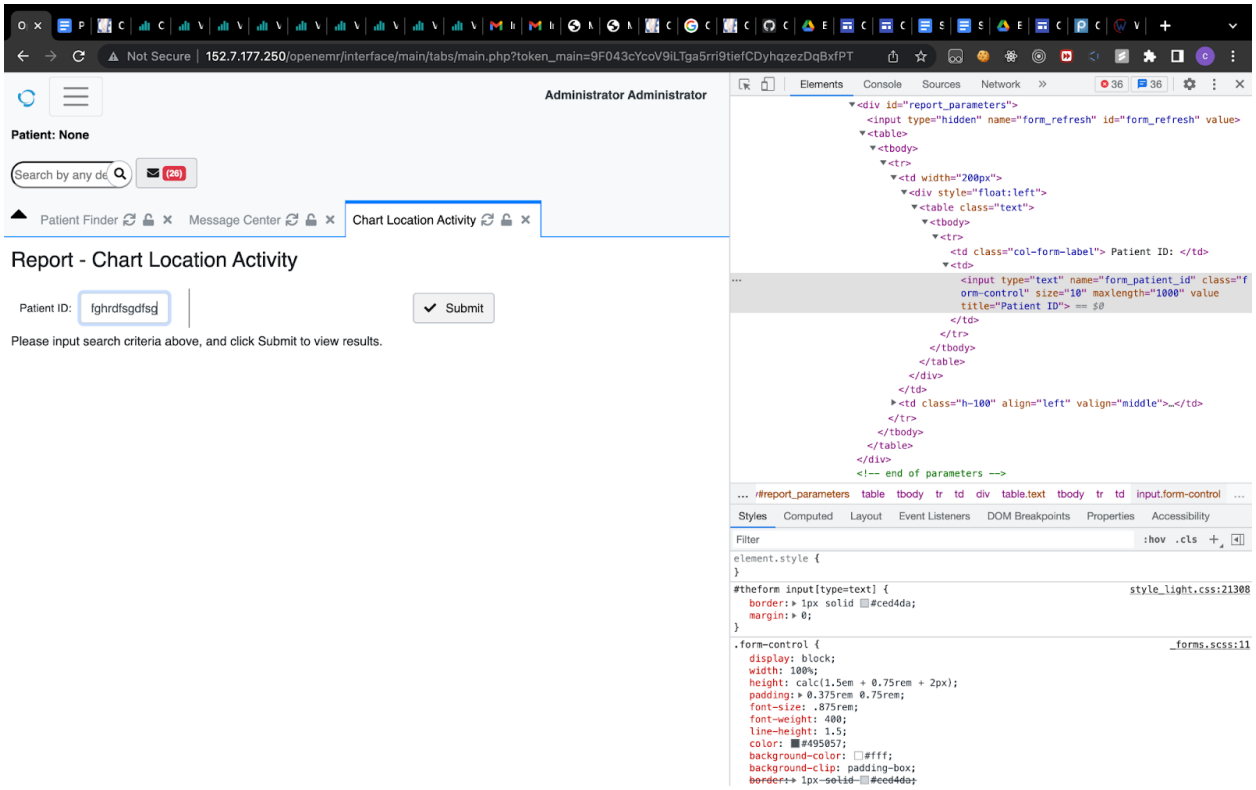
Expected Results:
● The given request should be rejected when tampered with some random input that does not meet our validation requirements.

CWE Info: 1029 OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure

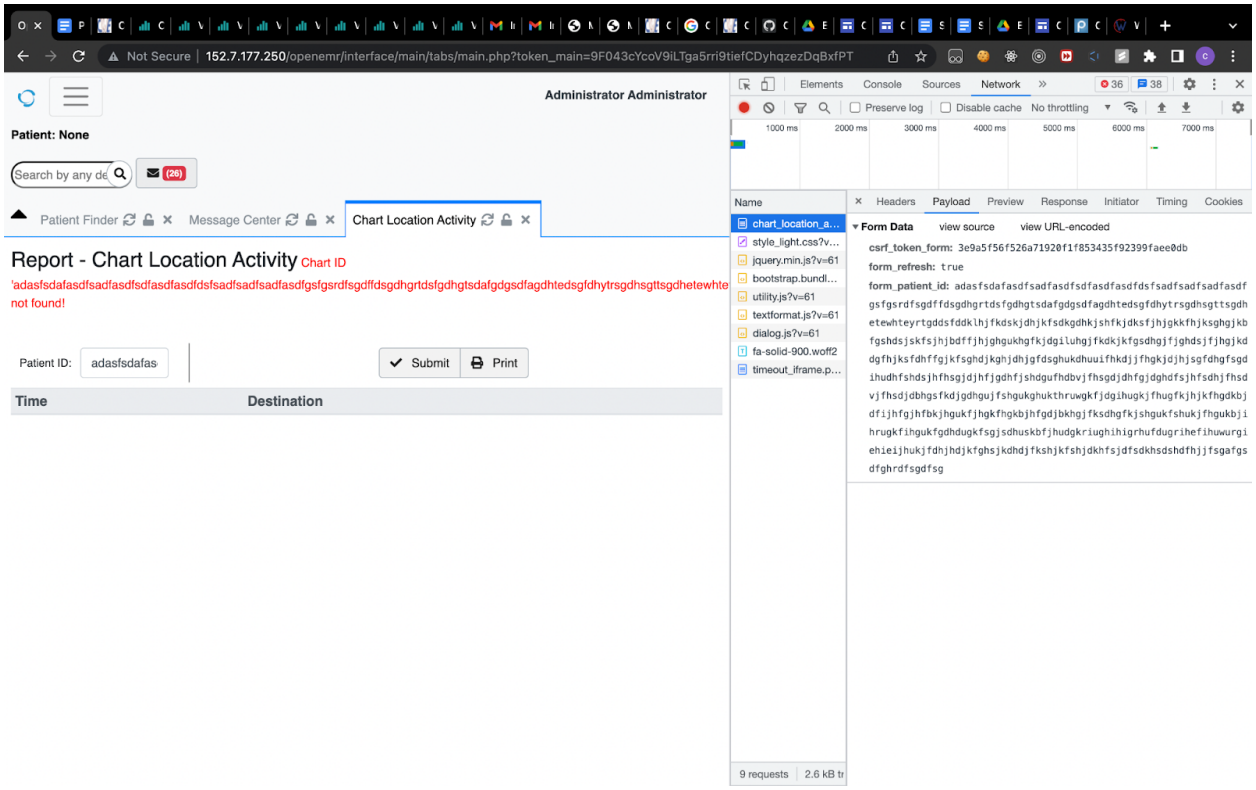
9. Test Case ID: 1.5.3

Description: Verify that input validation is enforced on a trusted service layer.

Repeatable Steps:
1. Login as any user (Eg: username: admin password: pass)
2. In the top navigation bar, hover over the Reports tab then hover over the Visits tab and select Chart Activity.
3. Now you will see a Patient ID: field. Inspect it and make changes in the browser html as shown in the below screenshot.



4. Here you have modified the max length to 1000 and now add some random lengthy input text to the patient ID.
5. Once the changes are done navigate to the network tab and clear the network calls already stacked.
6. Once the network tab is empty click on the Submit button.
7. Now you should be able to see a POST call being made to the backend with the tampered input value which we have given as shown in the below screenshot.



Expected Results:

● The given request should be rejected when tampered with some random input that does not meet our validation requirements. Because it might lead to a memory overflow.

CWE Info: 602 Client-Side Enforcement of Server-Side Security.

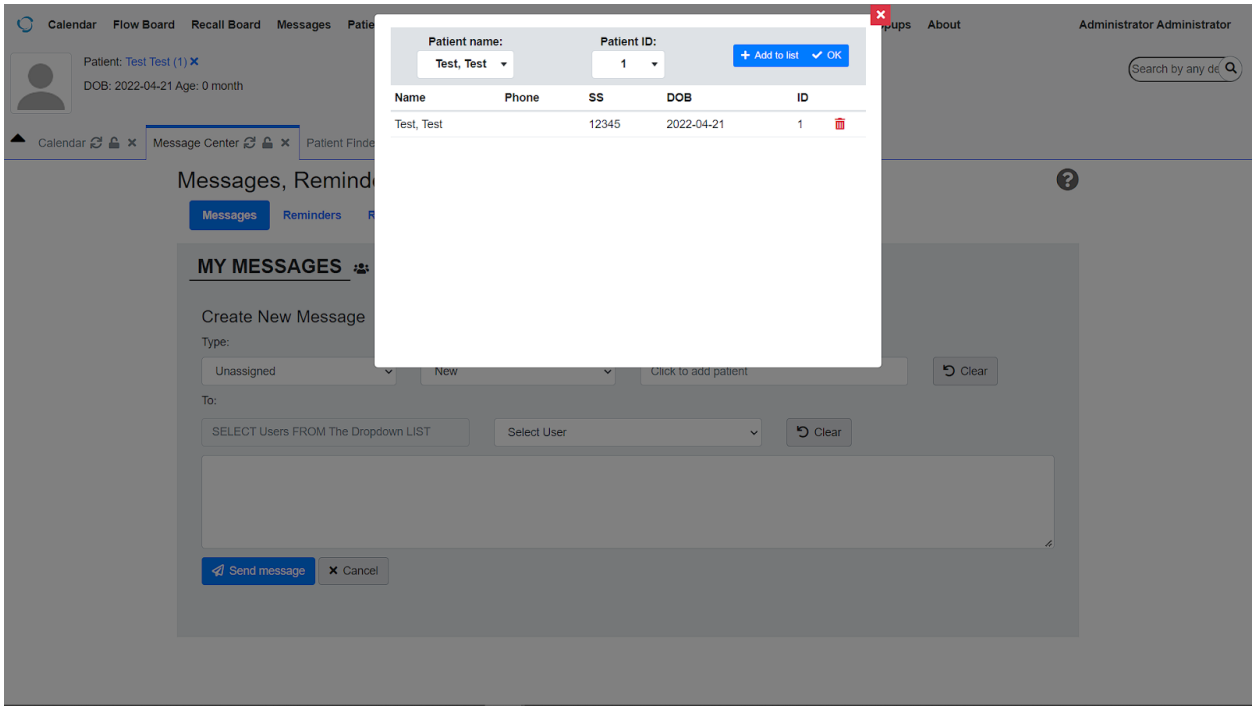
# Vulnerability	Elapsed Time	Ref Info for Video Traceability	CWE	Commentary
1	About 3 minutes	2hr 42min	1029	Video gives a rough idea on vulnerability for more detailed view use the repeatable steps mentioned above.
2	About 3 minutes	2hr 53min 30seconds	602	Video gives a rough idea on vulnerability for

				more detailed view use the repeatable steps mentioned above.

10. Test Case ID: 8.3.4-1

Description: Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data.

Repeatable Step:
1. Launch the openemr application in vcl.
2. Open http://localhost/openemr in Firefox browser within vcl.
3. Login as admin. (username: admin_openemr Password: admin_openemr)
4. On the top header, hover over Patient/Client and click on Patients.
5. Click on “Add New Patient” button.
6. Enter Information First Name: Test, Last Name: Test, DOB: 4/20/2022, Sex: M, SS: 12345
7. Click on Create New Patient.
8. Click on Confirm Create New Patient.
9. Click past any new patient creation popups
10. Click on the Message Center tab.
11. Select Messages, then click Add New
12. Click on the Patient box
13. On the Patient Name dropdown, select Test, Test
14. Click Add to list
15. View patient Test, Test’s information



Expected Results:

- The patient's social security number should not be displayed to all users who send them a message

CWE Info:

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Vulnerability #	Elapsed Time	Ref Info for Video Traceability	CWE	Commentary
1	2 minutes	13m 30s	200	Video gives description and confirmation of SSN being displayed in patient selection screen when sending a message

5. Vulnerability discovery efficiency and effectiveness

Technique	# of True Positives	Total time (hours)	Efficiency: # vulnerabilities / total. time	Detecting Exploitable vulnerabilities ? (High/Med/Low)	Unique CWE numbers
Manual black box	7	8	0.875	Medium	598, 614, 419, 548, 138, 400, 134
Static Application Security Testing (SAST)	17	17.5	0.971	Medium	200, 20, 73, 359, 326, 918, 117, 601, 147, 95, 1004, 798, 295, 22, 829, 338
Dynamic Application Security Testing (DAST)	14	27	0.518	Low	89, 602, 20, 359, 548, 540, 346
Interactive Application Security Testing (IAST)	5	4	1.25	High	1029, 1004, 614, 436, 548
Penetration testing	10	12	0.833	Medium	1029, 602,120, 209,434, 79, 200

Reflection:
As you can observe from the above table, SAST has the scope of finding and pinpointing the exact vulnerabilities within the source code but the amount of time required to analyze the report is high. DAST though it detects application server misconfiguration and other environmental vulnerabilities precisely, it produces a lot of false positives and the exact issue is not clearly identified as it is only dealing at the application level. IAST has the highest efficiency as it combines both the advantages of SAST and DAST, the chances of finding both implementation and design bugs are very

high with this tool. The efficiency of black-box testing and penetration testing is a bit low compared to SAST and IAST, but higher than DAST. This is what we expected to happen since SAST and IAST are automated tools and can work faster than a normal human. Whereas in the case of DAST tools like ZAP, we have to find certain places where we want to explore SQL Injection and XSS vulnerabilities, and ZAP used to take a long time to run through the list of inputs for each type.

As shown in the above table, the tools find a wide range of CWE's that are truly exploitable. Among them it can be observed that there are few that are detected in mostly all the tools like CWE - 20 and CWE - 602 which lack sufficient input validation and API validation that leads to client side bypassing and other attacks. We can also see another common vulnerability, CWE - 548 which is exposure of information through directory listing detected by all the tools, as this is a common server configuration error that is done. Each tool has different patterns and methodologies in finding the vulnerabilities which makes them unique but by filtering the true positive issues generated from the tool's report, we can find the subset of CWE's that are most likely to be detected(If such a vulnerability exists in the application).