1. ZAP

Client-side bypassing

1. V5.3 Output encoding and Injection Prevention

Test Case id: 5.3.4-1

Description: Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.

Repeatable Steps:

1. Open ZAP in vcl.
2. Click on Manual Explore.
3. Enter the openemr URL "**http://localhost/openemr**" in the URL dialogue box, and then click on launch browser with Firefox as the default browser.
4. Login as administrator in firefox browser with username: **admin_openemr** and password: **admin_openemr**
5. In ZAP, there is a tab towards the left hand side. Under Sites, expand http://localhost, openemr, interface, main.
6. Under main directory, you can find a POST:main_screen.php file.
7. Right click on POST:main_screen.php file and select Break, and click on save.
8. In firefox browser, logout and login as administrator again with exact credentials. Username: **admin_openemr** and password: **admin_openemr**
9. You can see a HTTP Request dialogue box.
10. Go back to the ZAP application. You will be able to see that application stopped at the POST end-point since we added the break point.
11. Click on **Request** on top header of the application.
12. In the dialogue box below (Break tab), you'll be able to see the administrator credentials. Update authUser=admin_openemr to **' union (select @@version) --**
13. Click on the play icon to forward through on top of the ZAP application.
14. Go back to firefox browser to see the HTTP response dialogue box.
15. Click on continue in the firefox browser HTTP window.

Expected Results: The login attempt should fail displaying an error message saying "Invalid username or password". No database version information is displayed on the screen.

CWE: 89, Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

2. V5.3 Output encoding and Injection Prevention

Test Case id: 5.3.4-2

Description: Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.

Repeatable Steps:

1. Open ZAP in vcl.
2. Clock on Manual Explore.
3. Enter the openemr url "**http://localhost/openemr**" in the URL dialogue box, and then click on launch browser with Firefox as default browser.
4. Login as administrator in firefox browser with username: **admin_openemr** and password: **admin_openemr**
5. On the top headers, hover over **Patient/Client** and click on **Patients.**
6. Under Patient finder, click on Add New Patient.
7. Create a new patient. Select Mr. from pronouns drop down. Type **Abc** as first name in first box, leave out second box and type **def** as last name in third box.
8. Select DOB as March 07 2022.
9. Select Sex as **Male.**
10. Scroll down and click on **Create New Patient.**
11. Note: Sometimes you'll not be able to see Create New Patient in firefox browser. Click on green icon on bottom right corner to hide history details.
12. In ZAP, there is a tab towards the left hand side. Under Sites, expand http://localhost, openemr, interface, new
13. Under new directory, you can find a POST:new_comprehensive_save.php(csrf token….)
14. The URL in request will be **http://localhost/openemr/interface/new/new_comprehensive_save.php**
15. Right click on POST:new_comprehensive_save.php file and select Break, and click on save.
16. Go back to the firefox browser and repeat steps 5,6,7,8,9,10.

17. When you click on Create New Patient in step 10, you'll be able to see a HTTP request dialogue box.
18. Come back to ZAP application to find that the application stopped at the break point.
19. In the dialogue box below (Break tab), you'll be able to see the **form_DOB** in the payload. Update form_DOB=03-07-2022 to **form_DOB=' or 1=1–**
20. Click on the play icon to forward through on top of the ZAP application.
21. Go back to firefox browser to see the HTTP response dialogue box.
22. Click on continue in the firefox browser HTTP window.
23. If any API error window pops up, close error window.
24. On the top headers, hover over **Patient/Client** and click on **Patients** to see the new Patient details created just now.

Expected Results: The new patient should not be created.

CWE: 89, Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

3. V4.1 General Access Control Design

Test Case id: 4.1.1-1

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

Repeatable steps:

1. Open ZAP in vcl.
2. Clock on Manual Explore.
3. Enter the openemr URL "**http://localhost/openemr**" in the URL dialogue box, and then click on launch browser with Firefox as the default browser.
4. log in as administrator in firefox browser with username: **admin_openemr** and password: **admin_openemr**
5. On the top headers, hover over **Patient/Client** and click on **Patients.**
6. Under Patient finder, click on Add New Patient.
7. Create a new patient. Select Mr. from pronouns drop down. Type **Abc** as the first name in the first box, leave out the second box, and type **def** as the last name in the third box.
8. Select DOB as March 07, 2022.

9. Select Sex as **Male.**
10. Scroll down and click on **Create New Patient.**
11. Note: Sometimes you'll not be able to see Create New Patient in the firefox browser. Click on the green icon on the bottom right corner to hide history details.
12. On the top headers click on **Messages.**
13. Under Messages, Reminders, Recalls click on **Add New.**
14. Select **Chart Note** under Type.
15. Click on the Patient box and select **Def,abc** from the Patient name drop down.
16. Click on Add to list, and click on ok.
17. Under To: select **Administrator, Administrator**
18. Type a message as **hello** in the message field.
19. Click on send message.
20. In ZAP, there is a tab towards the left-hand side. Under Sites, expand http://localhost, openemr, interface, main, messages.
21. Under the orders directory, you can find a POST:messages.php(begin, form_active, showall, sort by, sort order)
22. The URL in request will be **http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1**
23. Right-click on POST:messages.php file and select Break, and click on save.
24. Go back to the firefox browser and repeat steps 12,13,14,15,16,17,18,19.
25. When you click on send message in step 19, you'll be able to see a HTTP request dialogue box.
26. Come back to the ZAP application to find that the application stopped at the break point.
27. In the dialogue box below (Break tab), you'll be able to see the **form_note_type** in the payload. Update form_note_type=Chart+Note to **form_note_type=<script>alert('xss')</script>**
28. Click on the play icon to forward through on top of the ZAP application.
29. Go back to the firefox browser to see the HTTP response dialogue box.
30. Click on continue in the firefox browser HTTP window.
31. If any API error window pops up, close the error window.
32. Close the messages dialog box and on top of the web page click on messages again to refresh the messages window.

Expected Results: Since the new message form_note_type contains malicious or invalid characters, the new message should not be stored and sent.

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

4. V5.1 Input Validation

Test Case id: 5.1.3-1

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

Repeatable Steps:

1. Open ZAP in vcl.
2. Clock on Manual Explore.
3. Enter the openemr URL "**http://localhost/openemr**" in the URL dialogue box, and then click on launch browser with Firefox as the default browser.
4. log in as administrator in firefox browser with username: **admin_openemr** and password: **admin_openemr**
5. On the top headers, hover over **Patient/Client** and click on **Patients.**
6. Under Patient finder, click on Add New Patients.
7. Create a new patient. Select Mr. from pronouns drop down. Type **test** as the first name in the first box, leave out the second box, and type **test** as the last name in the third box
8. Select DOB as June 5th, 1975.
9. Select Sex as **Male.**
10. Scroll down and click on **Create New Patient.**
11. Close the success patient created dialogue box.
12. Click on patient finder and the patient we just created , (test, test)
13. On bottom right, Under Recurrent Appointments, click on **Edit** Allergies
14. Click on **Add** allergies, select **penicillin** as the title and select the begin date as **2022-03-08** and Reaction as **Nausea** click on **save** allergy
15. Note: Sometimes you'll not be able to see Create New allergy for a patient in firefox browser. Click on green icon on bottom right corner to hide history details.
16. In ZAP, there is a tab towards the left hand side. Under Sites, expand http://localhost, openemr, interface, patient_file, summary
17. Under this directory you can find POST:add_edit_issue.php entry, right click that and click on break and save it.
18. Under Request tab, the URL will be **http://localhost/openemr/interface/patient_file/summary/add_edit_issue.php?issue=0&thistype=allergy**

19. Repeat the steps from 12-14 after click on save allergy, we can change the actual input (In break tab) for the following input **form_reaction** in the response tab to series of character ‘a’ of length 1500 (Buffer Overflow)
20. Click on the play icon to forward through on top of the ZAP application.
21. Go back to firefox browser to see the HTTP response dialogue box.
22. Click on continue in the firefox browser HTTP window.
23. On patient Issues tab, we can see the list of allergies, in the allergy we created we can check that Reaction is series of character ‘a’ which is not part of our options in the select element.

Expected Results: The Allergy for the corresponding patient **test**, should **not** be created.

CWE: 20, Improper Input Validation

5. V5.1 Input Validation

Test Case id: 5.1.3-2

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

Repeatable steps:

1. Open ZAP in vcl.
2. Clock on Manual Explore.
3. Enter the openemr url “**http://localhost/openemr**” in the URL dialogue box, and then click on launch browser with Firefox as default browser.
4. Login as administrator in firefox browser with username: **admin_openemr** and password: **admin_openemr**
5. On the top headers, hover over **Procedures** and click on **Batch Results.**
6. Click on the field corresponding to the “Procedure:”. You will be popped up with a form containing heading as “**Configure Orders and Results**”
7. Click on the “**+Add Top Level**” button and you will be popped up with a form.
8. In that form fill the values as “Procedure Tier: Group”, “Name: dummy name”, “Description: dummy” and “Sequence: dummy” and click Save.
9. In ZAP, there is a tab towards the left-hand side. Under Sites, expand http://localhost, openemr, interface, orders.

10. Under the orders directory, you can find a POST:types_edit.php(parent, typeid)(form_body_site, form_description, form_diagnosis_code, form_laterality, form_name, form_procedure_code, form_procedure_type, from_range, form_realted_code, from_route_admin, form_save, form_seq, form_specimen, form_standard_code, form_units)
11. The URL in request will be "**https://localhost/openemr/interface/orders/types_edit.php?typeid=0&parent=0**" .
12. Right click on the URL mentioned in the step 10, and select Break, and click on save.
13. Go back to the firefox browser and repeat steps 5, 6, 7, 8.
14. When you click on Save this time as mentioned in the step 8, you might be able to see a HTTP request dialogue box.
15. Come back to the ZAP application to find that the application stopped at the break point.
16. In the dialogue box below (Break tab), you'll be able to see the **form_procedure_type** in the payload. Update form_procedure_type value with **form_procedure_type=' ; drop table temp --**
17. Click on the play icon to forward through on top of the ZAP application.
18. Go back to the firefox browser to see the HTTP response dialogue box if present otherwise no need to worry about it.
19. Click on continue in the firefox browser HTTP window.
20. If any API error window pops up, close the error window.
21. Close all the dialog boxes and on top of the web page and logout and login to the application again, on top of the headers hover over **Procedures** and click on **Batch Results** to navigate again to the same page and click on the field corresponding to "Procedure:" and check for the "Category" field value by clicking on the Edit(pencil) button of the row containing your order.

Expected Results: Since the new order form_procedure_type contains malicious or invalid characters, the new message should not be stored and sent to the backend.

CWE: 20, Improper Input Validation

Total time spent on Black Box Test Cases:

1. Individually we spent 2hours each planning and running the black-box test cases. So a total of 8hours was spent on black-box test cases.
2. Out of 5 black-box test cases, we found 4 vulnerabilities.
3. So, we found 1 true positive per hour.

Fuzzing

1. Ruleset Used: SQL Injection
Location: Creating a new appointment
URL: method: POST, end-point:
http://localhost/openemr/interface/main/calendar/add_edit_event.php

Results: True Positive, for **form_patient=' or 1=1 –**, the new meeting record is saved with no patient information

Test Case ID: 5.3.4-1

Description: Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.

Preconditions:

- A patient exists on this instance of OpenEMR
- If a patient does not exist, follow these steps to create one:
    - Move the cursor over **Patient/Clinic** and select **New/Search** from the dropdown
    - Give the patient a name (e.g. Test, Test), a DOB, and Sex
    - Select **Create New Patient** on the bottom of the screen

Repeatable Steps:

1. In VCL, Open OWASP ZAP and click on the “quick start” browser that is automatically configured to use the ZAP proxy. (a small firefox icon on top of the ZAP application).
2. Make sure that you install the Firefox browser.
3. Navigate to the openemr webpage in firefox. http://localhost/openemr
4. Login as admin with username: admin_openemr and password: admin_openemr
5. Under the **Calendar** tab, select the ‘+’ icon to create a new appointment
6. Click the patient field and search for any patient (e.g. test)
7. Add this patient to the meeting
8. Note the date and time of the appointment
9. Click Save. If a popup appears saying the provider is not available, click OK
10. Go back to the ZAP application and open the History tab at the bottom of the application.
11. Find the POST endpoint with URL:
    http://localhost/openemr/interface/main/calendar/add_edit_event.php?eid=
12. In the Request tab, select the form_patient value.
13. Right-click on the form_patient value and select fuzz.
14. Select Payloads from the Fuzzer window.
15. Click on Add.
16. Select File Fuzzers as Type.
17. Expand jbrofuzz and select SQL Injection.
18. Click on Add and then OK to return back to the Fuzzer window.

19. Click Start Fuzzer to begin the fuzzing process.
20. Go through the results and find the payload with form_patient=’ or 1=1–
21. Go back to OpenEMR and select the date on the calendar for which the original appointment was made.

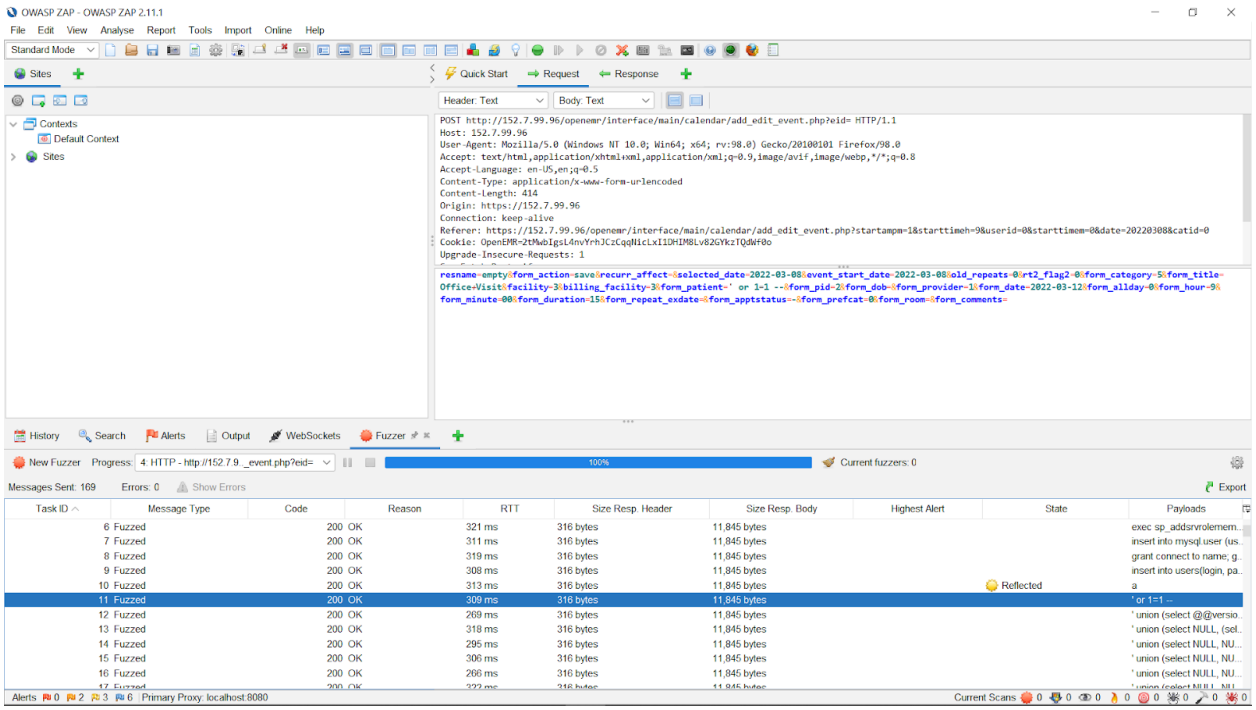
Expected Results: The new appointment should not be created since the patient field contains malicious code.

CWE: 89 - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Mitigation techniques to fix this issue:

- Input validation is not available on the server-side. Since DOB is a calendar date, we need to make sure that there are no special characters except the hyphen in the DOB input.
- Add an allow list for the DOB input validation like regex.
- Use prepared statements so that they can sanitize the input while executing SQL queries.
- Using database frameworks like Hibernate would mitigate this issue.

Screenshot of ZAP:



2. Ruleset Used: SQL Injection
Location: While creating a new Patient.
URL: method: POST, end-point:
http://localhost/openemr/interface/new/new_comprehensive_save.php

Results: True positive, for **form_DOB=' or 1=1–**, the new patient record will be saved with DOB=0000-00-00.

Test Case id: 5.3.4-2

Description: Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.

Repeatable steps:

1. In VCL, Open OWASP ZAP and click on the "quick start" browser that is automatically configured to use the ZAP proxy. (a small firefox icon on top of the ZAP application).
2. Make sure that you install the Firefox browser.
3. Navigate to the openemr webpage in firefox. http://localhost/openemr
4. Login as admin with username: admin_openemr and password: admin_openemr
5. On top headers, hover on **Patient/Client** and click on Patients.
6. Under Patient Finder click on **Add New Patient**
7. Enter Name. Select Mr. from the dropdown. Enter abc in the first box and def in the last box (middlebox can be left empty).
8. Select DOB as 03/04/2022 from the calendar.
9. Select Sex as Male from the dropdown.
10. Scroll down the page and click on Create New Patient.
11. Go back to the ZAP application and open the History tab at the bottom of the application.
12. Find the POST endpoint with URL:
    http://localhost/openemr/interface/new/new_comprehensive_save.php
13. In the Request tab, select the form_DOB value.
14. Right-click on the form_DOB value and select fuzz.
15. Select Payloads from the Fuzzer window.
16. Click on Add.
17. Select File Fuzzers as Type.
18. Expand jbrofuzz and select SQL Injection.
19. Click on Add and then OK to return back to the Fuzzer window.
20. Click Start Fuzzer to begin the fuzzing process.
21. Go through the results and find the payload with form_DOB=' or 1=1–
22. Go back to OpenEMR and hover on Patient/Client on the top headers and click on Patients.

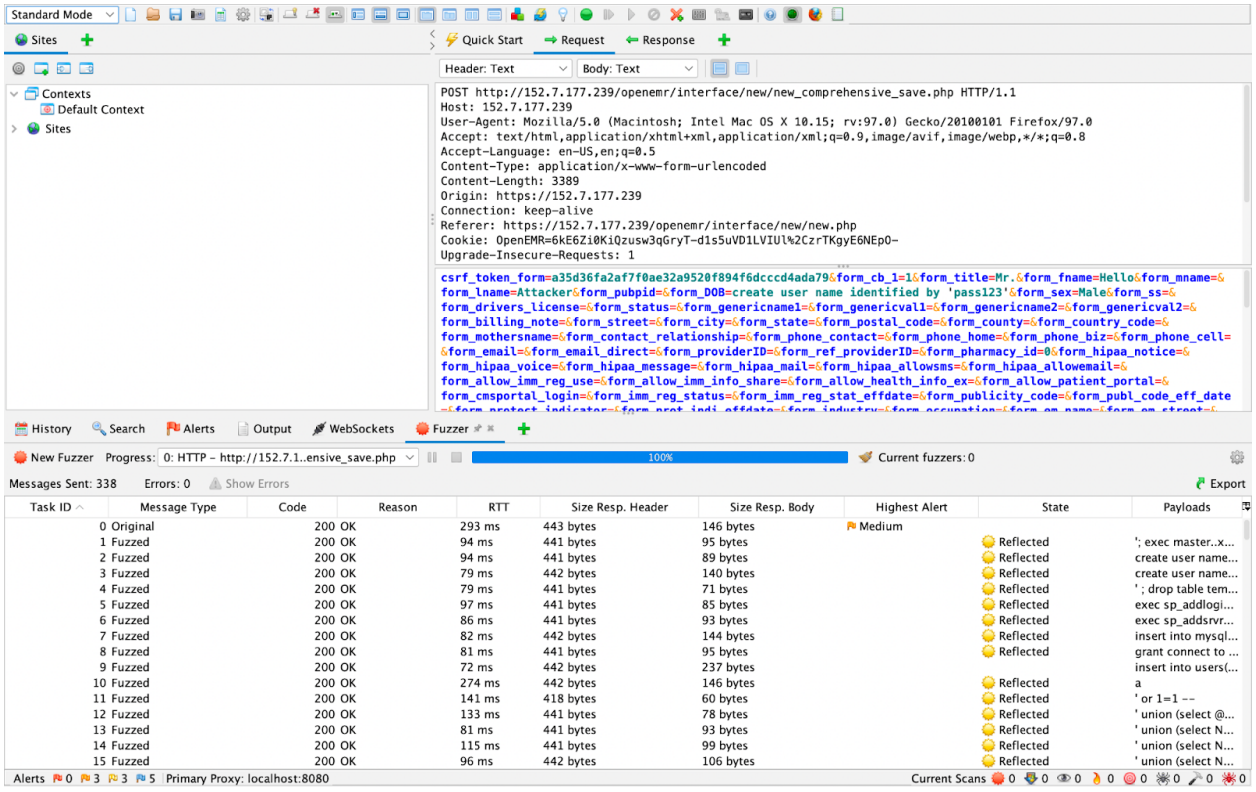
Expected Results: The new patient should not be created since the DOB field contains malicious code.

CWE: 89, Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Mitigation techniques to fix this issue:

1. Input validation is not available on the server-side. Since DOB is a calendar date, we need to make sure that there are no special characters except the hyphen in the DOB input.
2. Add an allow list for the DOB input validation like regex.
3. Use prepared statements so that they can sanitize the input while executing SQL queries.
4. Using database frameworks like Hibernate would mitigate this issue.

Screenshot of ZAP:



3. Ruleset Used: XSS
Location: Creating a new message
URL: method: POST, end-point: http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1

Results: True positive, for **form_note_type=<script>alert(`xss`)<script>**, a new message is saved and sent to a patient with a malicious script. The malicious script can be seen under the Type in the messages page.

Test Case id: 4.1.1-1

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

Repeatable Steps:

1. In VCL, Open OWASP ZAP and click on the “quick start” browser that is automatically configured to use the ZAP proxy. (a small firefox icon on top of the ZAP application).
2. Make sure that you install the Firefox browser.
3. Navigate to the openemr webpage in firefox. http://localhost/openemr
4. log in as admin with username: admin_openemr and password: admin_openemr
5. Create a new Patient. To do this hover over Patient/Client on top headers and click on patients. Under Patient Finder click on **Add New Patient**
6. Enter Name. Select Mr. from the dropdown. Enter abc in the first box and def in the last box (the middlebox can be left empty).
7. Select DOB as 03/04/2022 from the calendar.
8. Select Sex as Male from the dropdown.
9. Scroll down the page and click on Create New Patient.
10. On the top header click on **Messages.**
11. Under My messages click on Add new
12. Under Create New Message select Type as Chart Note.
13. Click on the Patient box and select **def,abc** from the Patient name dropdown.
14. Click on Add to list, and click on ok.
15. Under To: select **Administrator, Administrator**
16. Type a message as **hello** in the message field.
17. Click on send message.
18. Go back to the ZAP application and open the History tab at the bottom of the application.
19. Find the POST endpoint with URL: http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1
20. In the Request tab, select the form_note_type value.
21. Right-click on the form_note_type value and select fuzz.
22. Select Payloads from the Fuzzer window.
23. Click on Add.
24. Select File Fuzzers as Type.
25. Expand jbrofuzz and select XSS.
26. Click on Add and then OK to return back to the Fuzzer window.
27. Click Start Fuzzer to begin the fuzzing process.

28. Go through the results and find the payload with form_note_type=<script>alert(`xss`)</script>
29. Go back to openemr and click on messages.
30. You can see a list of messages since fuzzing uses a set of payloads.
31. Find the message with type as <script>alert(`xss`)</script>

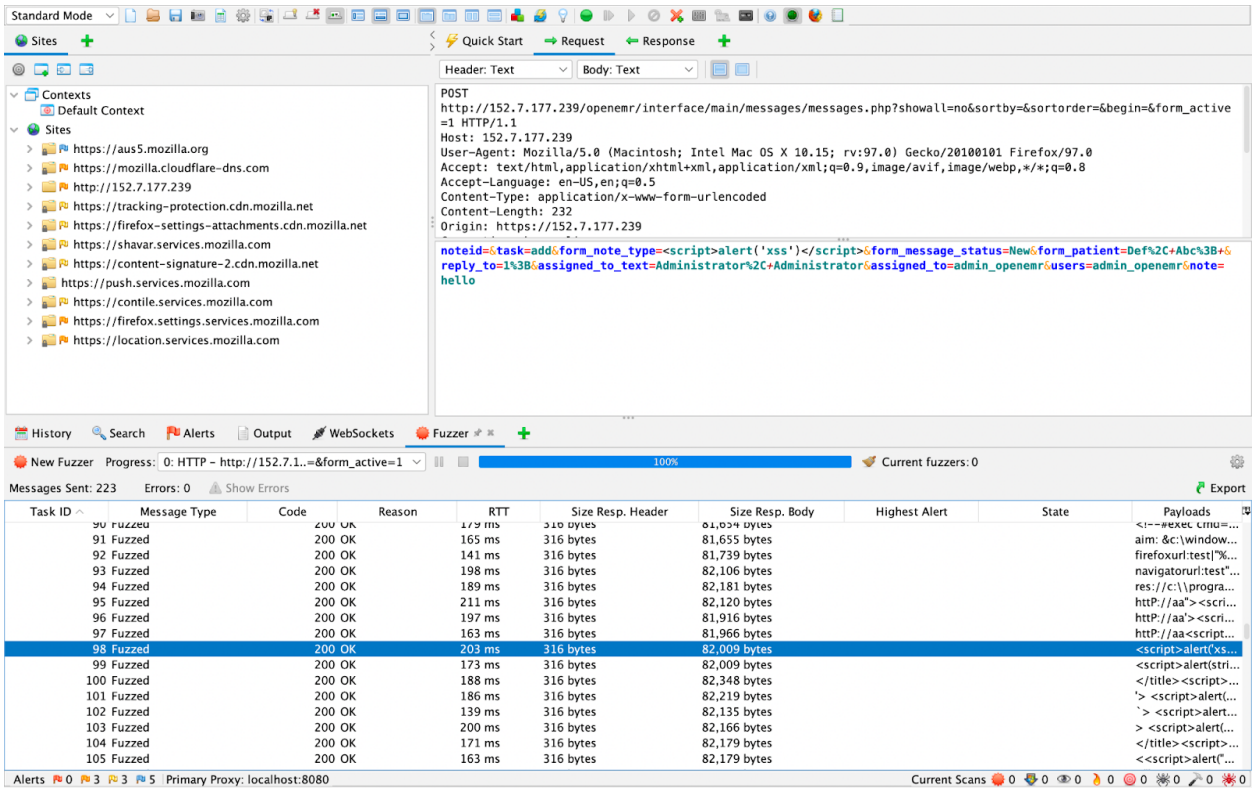
Expected Results: Since the new message form_note_type contains malicious or invalid characters, the new message should not be stored and sent.

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

Mitigation techniques to fix this issue:

1. On the client-side, since it's a dropdown, we have to make sure input validation is present on every field on the client-side and server-side.
2. Use deny list/allow list to validate the input on the server-side.
3. Can use output encoding libraries like Microsoft's Anti-XSS library.

Screenshot of ZAP:



4. Ruleset Used: Buffer Overflow
Location: Creating a new Allergy
URL: method: POST, end-point:
http://localhost/openemr/interface/patient_file/summary/add_edit_issue.php?issue=0&thistype=allergy

Results: True Positive, the series of characters 'a' should not be accepted in the input field form_reaction while creating a new allergy for a particular patient

Test Case id: 5.1.3-1

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

Repeatable Steps:

1. In VCL, Open OWASP ZAP and click on the "quick start" browser that is automatically configured to use the ZAP proxy. (a small firefox icon on top of the ZAP application).
2. Make sure that you install the Firefox browser.
3. Navigate to the openemr webpage in firefox. http://localhost/openemr
4. log in as admin with username: admin_openemr and password: admin_openemr
5. Create a new Patient. To do this hover over Patient/Client on top headers and click on patients. Under Patient Finder click on **Add New Patient**
6. Create a new patient. Select Mr. from pronouns drop down. Type **test** as the first name in the first box, leave out the second box, and type **test** as the last name in the third box
7. Select DOB as June 5th, 1975.
8. Select Sex as **Male.**
9. Scroll down and click on **Create New Patient.**
10. Close the success patient created dialogue box.
11. Click on patient finder and the patient we just created , (test, test)
12. On bottom right, Under Recurrent Appointments, click on **Edit** Allergies
13. Click on **Add** allergies, select **penicillin** as the title and select the begin date as **2022-03-08** and Reaction as **Nausea** click on **save** allergy
14. Go back to the ZAP application and open the History tab at the bottom of the application.
15. Find the POST endpoint with URL: **http://localhost/openemr/interface/patient_file/summary/add_edit_issue.php?issue=0&thistype=allergy**
16. In the request tab, select form_reaction and right click and select fuzz
17. Select Payloads from the Fuzzer window.
18. Click on Add.

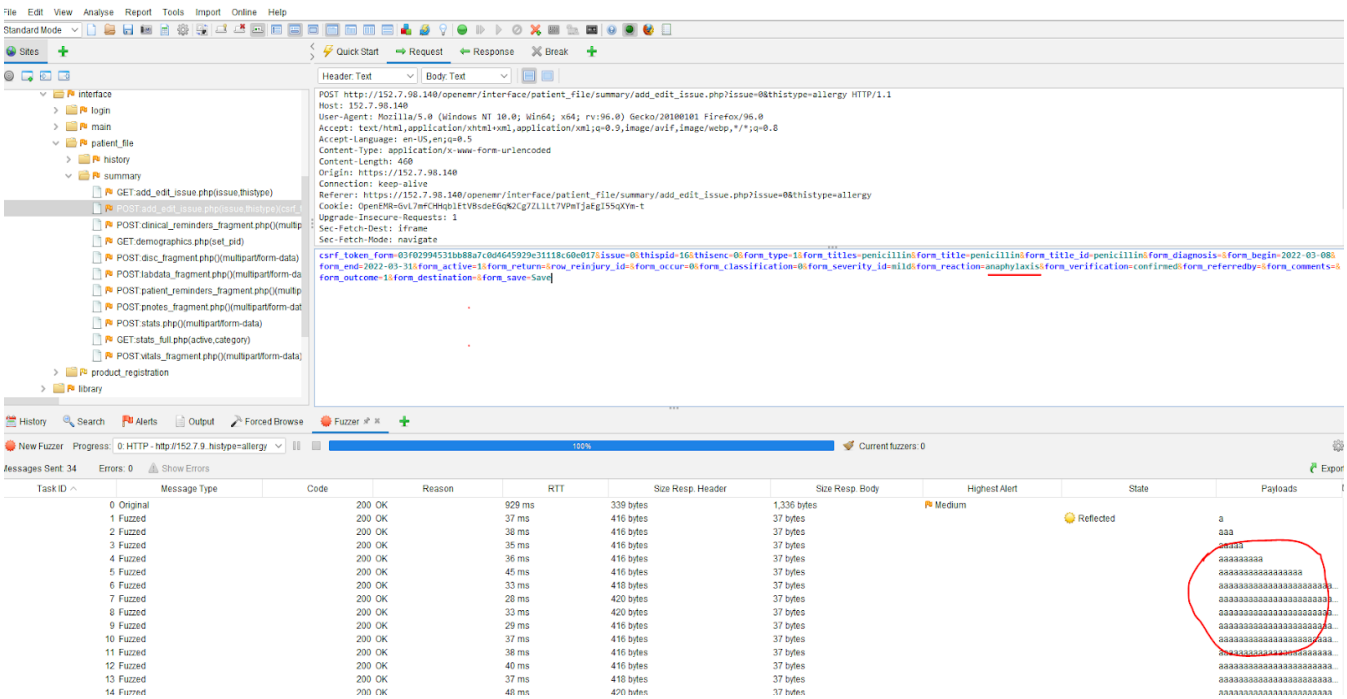
19. Select File Fuzzers as Type.
20. Expand jbrofuzz and select Buffer Overflow
21. Click on Add and then OK to return back to the Fuzzer window.
22. Click Start Fuzzer to begin the fuzzing process.
23. Go through the results and find the payload with series of characters ‘a’ of length 1500
24. Go back to openemr and click on allergies in patients tab
25. We can see a new entry in allergies tab, that has reaction as series of characters ‘a’

Expected Results: Since the new allergy record in form_reaction contains a value other than the given options in the select element, It should not be created / saved .

CWE: 20, Improper Input Validation

Mitigation techniques to fix this issue:

1. On the client-side, since it’s a dropdown, we have to make sure input validation is present on every field on the client-side, so that we do not submit form with invalid inputs.
2. We also need to limit the input length, this should be implemented both on client side and server side with API validation.



5. Ruleset Used: HTTP, XSS and SQL Injection
Location: Configure Orders and Results in Procedure Results tab.

URL: method: POST, end-point:
http://localhost/openemr/interface/orders/types_edit.php?typeid=7&parent=0

Results: True positive, for **form_procedure_type=<script>alert(`xss`)<script>**, a new procedure tier is saved and reflected in the frontend of the “Configure Order and Results” tab. The malicious script removes the existing procedure tier and introduces a new unknow type as “* *”.

Test Case id: 5.1.3-2

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

Repeatable Steps:

1. In VCL, Open OWASP ZAP and click on the “quick start” browser that is automatically configured to use the ZAP proxy. (a small firefox icon on top of the ZAP application).
2. Make sure that you install the Firefox browser.
3. Navigate to the openemr webpage in firefox. http://localhost/openemr
4. login as admin with username: admin_openemr and password: admin_openemr
5. On the top headers, hover over **Procedures** and click on **Batch Results.**
6. Click on the field corresponding to the “Procedure:”. You will be popped up with a form containing heading as “**Configure Orders and Results**”
7. Click on the “**+Add Top Level**” button and you will be popped up with a form.
8. In that form fill the values as “Procedure Tier: Group”, “Name: dummy name”, “Description: dummy” and “Sequence: dummy” and click Save.
9. In ZAP, there is a tab towards the left-hand side. Under Sites, expand http://localhost, openemr, interface, orders.
10. Under the orders directory, you can find a POST:types_edit.php(parent, typeid)(form_body_site, form_description, form_diagnosis_code, form_laterality, form_name, form_procedure_code, form_procedure_type, from_range, form_realted_code, from_route_admin, form_save, form_seq, form_specimen, form_standard_code, form_units)
11. The URL in request will be “**https://localhost/openemr/interface/orders/types_edit.php?typeid=0&parent=0**” .
12. Right click on the URL mentioned in the step 10, and select Break, and click on save.
13. Go back to the firefox browser and repeat steps 5, 6, 7, 8.
14. When you click on Save this time as mentioned in the step 8, you might be able to see a HTTP request dialogue box.
15. In the Request tab, select the form_procedure_type value.
16. Right-click on the form_note_type value and select fuzz.
17. Select Payloads from the Fuzzer window.
18. Click on Add.
19. Select File Fuzzers as Type.
20. Expand jbrofuzz and select XSS, HTTP and SQL Injection.

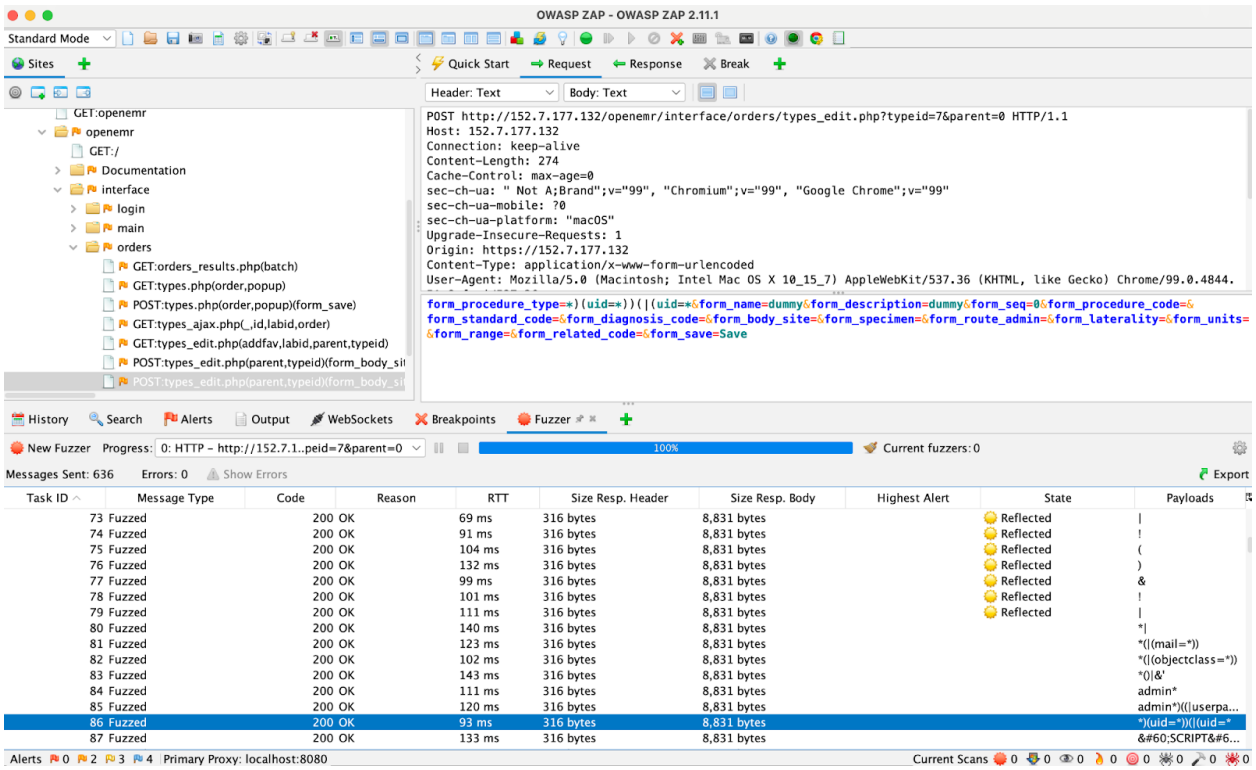
21. Click on Add and then OK to return back to the Fuzzer window.
22. Click Start Fuzzer to begin the fuzzing process.
23. Close all the dialog boxes and on top of the web page and logout and login to the application again, on top of the headers hover over **Procedures** and click on **Batch Results** to navigate again to the same page and click on the field corresponding to “Procedure:” and check for the “Category” field value by clicking on the Edit(pencil) button of the row containing your order.
24. Here in this case, when the fuzzing has happened with a different set of payloads, each payload tries to replace the existing procedure tier with a new one which cannot be properly rendered in the frontend, and gets reflected as “* *”.

Expected Results: Since the new order form_procedure_type contains malicious or invalid characters, the new message should not be stored and sent to the backend.

CWE: 20, Improper Input Validation

Mitigation techniques to fix this issue:

1. On the client-side, since it’s a dropdown, there must be proper input sanitization in-place to prevent malicious input being entered in that form field.
2. And also there must be proper checks on the server side before writing onto the database if there is malicious input. This will prevent malicious content from being rendered at the frontend when queried from the backend.



Total time spent on Fuzzing:

1. We spent 13 hours 30 minutes on this activity.
2. The ZAP scan took 30 minutes in total to run against all the test cases we planned.
3. We spent almost 10 hours analyzing and working with ZAP output. Initially, everything that ZAP tries out gave 200 responses. So, it took time for us to figure out a way how to catch the exact results.
4. To plan and run the black-box tests it took 3hours since we used the same black-box tests reported for Client-side bypassing.
5. So, we found 0.38 true positives per hour.

---

2. Vulnerable Dependencies

GitHub Checker:

Total: 51 Vulnerable dependencies

1. Regular Expression Denial of Service in postcss
CVE ID : CVE-2021-23382, CVE-2021-23368
Dependency: Transitive, Direct
Library: postcss
Safer Version: 8.2.13 or later
Severity: Moderate
Description: The package postcss versions before 7.0.36 or between 8.0.0 and 8.2.13 are vulnerable to Regular Expression Denial of Service (ReDoS) via getAnnotationURL() and loadAnnotation() in lib/previous-map.js.. . Transitive dependency of other libraries has the version 7.0.35 and The direct dependency of the application has the version 8.2.4.

2. Regular Expression Denial of Service in trim-newlines
CVE ID: CVE-2021-33623
Dependency: Direct, Transitive
Library: trim-newlines
Safer Version: 3.0.1 or later
Severity: High
Description: The trim-newlines package before 3.0.1 and 4.x before 4.0.1 for Node.js has an issue related to regular expression denial-of-service (ReDoS) for the .end() method. Openemr has a direct and transitive dependency with version V3.0.0

3. Exposure of Sensitive Information to an Unauthorized Actor in nanoid
CVE ID: CVE-2021-23566
Dependency: Direct and Transitive
Library: nanoid
Safer Version: 3.1.31 or later
Severity: Moderate
Description: The package nanoid before 3.1.31 are vulnerable to Information Exposure via the valueOf() function which allows to reproduce the last id generated. Here direct dependency is of the version 3.1.20 and there is a transitive dependency with postcss that required version to be 3.1.20 or later

4. Authentication Bypass in ADOdb/ADOdb
CVE ID: CVE-2021-3850
Dependency: Direct
Library: adodb/adodb-php
Safer Version: 5.20.21, 5.21.4 or later
Severity: Critical
Description: An attacker can inject values into a PostgreSQL connection string by providing a parameter surrounded by single quotes. Depending on how the library is used in the client software, this may allow an attacker to bypass the login process, gain access to the server's IP address, etc. Openemr is currently using version v5.20.18 directly and there is no transitive dependency here.

5. HTML comments vulnerability allowing to execute JavaScript code
CVE ID: CVE-2021-41165, CVE-2021-41164, CVE-2021-26272, CVE-2021-37695, CVE-2021-32809, CVE-2021-32808, CVE-2021-33829
Dependency: Direct
Library: ckeditor4
Safer Version: 4.17.0 or later
Severity: High
Description: A potential vulnerability has been discovered in CKEditor 4 HTML processing core module. The vulnerability allowed to inject malformed comments HTML bypassing content sanitization, which could result in executing JavaScript code. It affects all users using the CKEditor 4 at version < 4.17.0. Openemr is currently using version v4.15.0 directly and there is no transitive dependency here.

7. Remote code execution in zendframework and laminas-http
CVE ID: CVE-2021-3007, CVE-2022-23598
Dependency: Direct and Transitive
Library: laminas/laminas-http
Safer Version: 2.14.2 or later
Severity: Critical

Description: Zend Framework 3.0.0 has a deserialization vulnerability that can lead to remote code execution if the content is controllable, related to the __destruct method of the Zend\Http\Response\Stream class in Stream.php. Openemr currently uses version v2.14.0 as direct dependency and version v2.7 as transitive dependency.

8. Arbitrary File Creation/Overwrite via insufficient symlink protection due to directory cache poisoning using symbolic links
CVE ID: CVE-2021-37712, CVE-2021-32610, CVE-2021-37701, CVE-2021-32804, CVE-2021-32803, CVE-2020-36193
Dependency: Direct and Transitive
Library: tar
Safer Version: 4.4.18 or later
Severity: high
Description: Current version of tar, the logic was insufficient when extracting tar files that contained two directories and a symlink with names containing unicode values that normalized to the same value. Openemr currently uses v2.2.2 and tar-pack currently has transitive dependency on tar which is using V2.2.1

9. Prototype Pollution
CVE ID: CVE-2021-23413
Dependency: Direct and Transitive
Library: jszip
Safer Version:3.7.0 or later
Severity: Moderate
Description: This affects the package jszip before 3.7.0. Crafting a new zip file with filenames set to Object prototype values (e.g proto, toString, etc) results in a returned object with a modified prototype instance. Openemr uses has direct dependency with V3.5.0 and transitive with v3.2.0

10. Command Injection in lodash
CVE ID: CVE-2021-23337, CVE-2020-28500
Dependency: Direct and Transitive
Library: lodash
Safer Version: 4.17.21 or later
Severity: high
Description: lodash versions prior to 4.17.21 are vulnerable to Command Injection via the template function. Currently openemr using V4.17.20 as direct dependency but there are other libraries that are using V4.17.14, V4.17.15, V4.17.19

OWASP Dependency Check:

Total: 29 Vulnerable Dependencies

1. Inefficient Regular Expression Complexity in ansi-regex
CVE ID : CVE-2021-3807
Dependency: Transitive
Library: ansi-regex
Safer Version: 6.0.1 or later
Severity: High
Description: ansi-regex is vulnerable to Inefficient Regular Expression Complexity

2. Regular Expression Denial of Service in braces
CVE ID : CVE-2018-1109
Dependency: Transitive
Library: braces
Safer Version: 2.3.1 or later
Severity: Medium
Description: A vulnerability was found in Braces versions prior to 2.3.1. Affected versions of this package are vulnerable to Regular Expression Denial of Service (ReDoS) attacks.

3. Regular Expression Denial of Service in browserslist
CVE ID : CVE-2021-23364
Dependency: Transitive
Library: browserslist
Safer Version: 4.16.5 or later
Severity: Medium
Description: The package browserslist from 4.0.0 and before 4.16.5 are vulnerable to Regular Expression Denial of Service (ReDoS) during parsing of queries.

4. Improperly Controlled Modification of Dynamically-Determined Object Attributes in chart.js
CVE ID : CVE-2020-7746
Dependency: Transitive
Library: chart.js
Safer Version: 2.9.47 or later
Severity: High
Description: The options parameter is not properly sanitized when it is processed. When the options are processed, the existing options (or the defaults options) are deeply merged with provided options. However, during this operation, the keys of the object being set are not checked, leading to a prototype pollution.

5. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') in CKEditor4
CVE ID : CVE-2021-26272, CVE-2021-32808, CVE-2021-32809, CVE-2021-33829, CVE-2021-37695, CVE-2021-41164, CVE-2021-41165
Dependency: Transitive
Library: CKEditor4
Safer Version: 4.17.0 or later
Severity: Medium
Description: In the affected version a vulnerability has been discovered in the core HTML processing module and may affect all plugins used by CKEditor 4. The vulnerability allowed the injection of malformed HTML comments bypassing content sanitization, which could result in executing JavaScript code.

6. Prototype Pollution in copy-props
CVE ID : CVE-2020-28503
Dependency: Transitive
Library: copy-props
Safer Version: 2.0.5 or later
Severity: Critical
Description: The package copy-props before 2.0.5 are vulnerable to Prototype Pollution via the main functionality.

7. Cross Site Scripting & Prototype Pollution in datatables.net
CVE ID : CVE-2020-28458, CVE-2021-23445
Dependency: Transitive
Library: datatables.net
Safer Version: 1.11.3 or later for CVE-2021-23445, None for CVE-2020-28458
Severity: High
Description: CVE-2021-23445 affects the package datatables.net before 1.11.3. If an array is passed to the HTML escape entities function it would not have its contents escaped.
CVE-2020-28458 affects all versions of package datatables.net. These are vulnerable to Prototype Pollution due to an incomplete fix.

8. Uncontrolled Resource Consumption in glob-parent
CVE ID : CVE-2020-28469
Dependency: Transitive
Library: glob-parent
Safer Version: 5.1.2 or later
Severity: High
Description: The enclosure regex used to check for strings ending in enclosure containing path separator.

9. Uncontrolled Resource Consumption in hosted-git-info
CVE ID : CVE-2021-23362
Dependency: Transitive
Library: hosted-git-info
Safer Version: 3.0.8 or later
Severity: Medium
Description: The npm package `hosted-git-info` before 3.0.8 are vulnerable to Regular Expression Denial of Service (ReDoS) via regular expression shortcutMatch in the fromUrl function in index.js. The affected regular expression exhibits polynomial worst-case time complexity

10. Cross Site Scripting in jquery-1.10.2.min.js
CVE ID : CVE-2015-9251, CVE-2019-11358, CVE-2020-11022, CVE-2020-11023
Dependency: Direct
Library: jquery-1.10.2.min.js
Safer Version: 3.5.0 or later
Severity: Medium
Description: Passing HTML containing <option> elements from untrusted sources may execute untrusted code even after sanitizing it.

Comparison Report:

Dependency	OWASP CVE Count	GitHub CVE Count
adodb/adodb-php	0	1
ansi-regex	1	1
braces	3	2
browserslist	2	1
chart.js	1	1
ckeditor4	7	7
copy-props	1	1
datatables.net	2	2
glob-parent	2	1
hosted-git-info	3	1
jquery	45	1
jspdf	3	1
jszip	2	1
laminas/laminas-form	0	1
laminas/laminas-http	0	1
lodash	5	2
nanoid	1	1
path-parse	2	1
pear/archive_tar	0	2
phpmailer/phpmailer	0	3
phpoffice/phpspreadsheet	0	1
phpseclib/phpseclib	0	1
postcss	10	4
smarty/smarty	0	4
tar	8	5
trim-newlines	1	1
twig/twig	0	1
underscore	2	1
yargs-parser	0	1

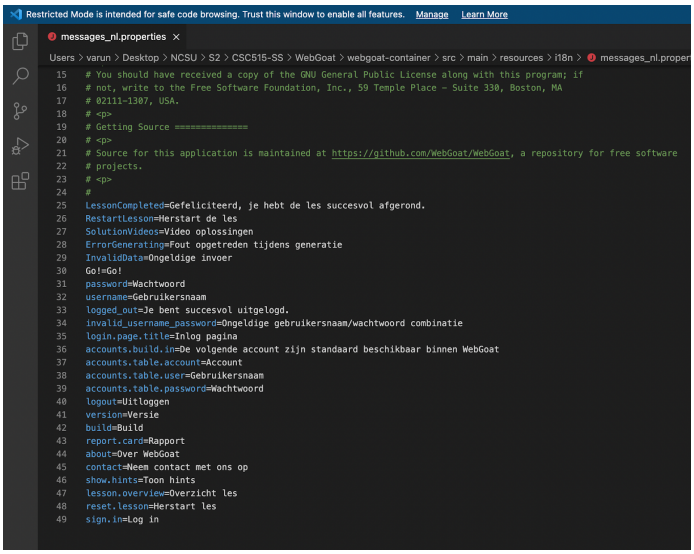
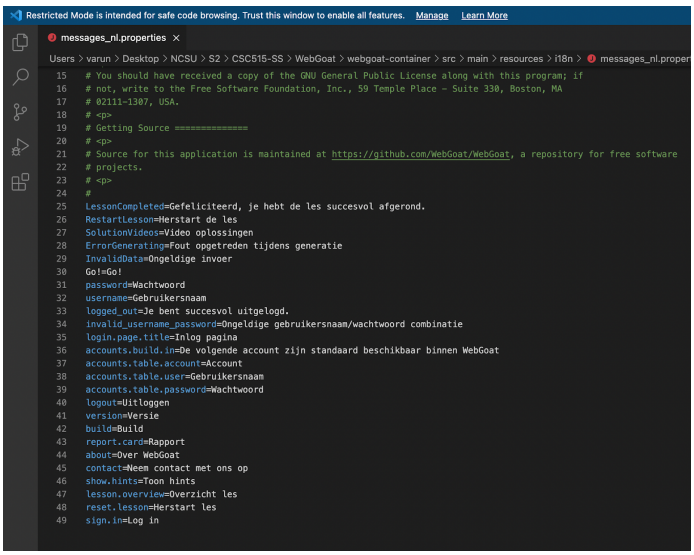
Explanation of Differing Results:

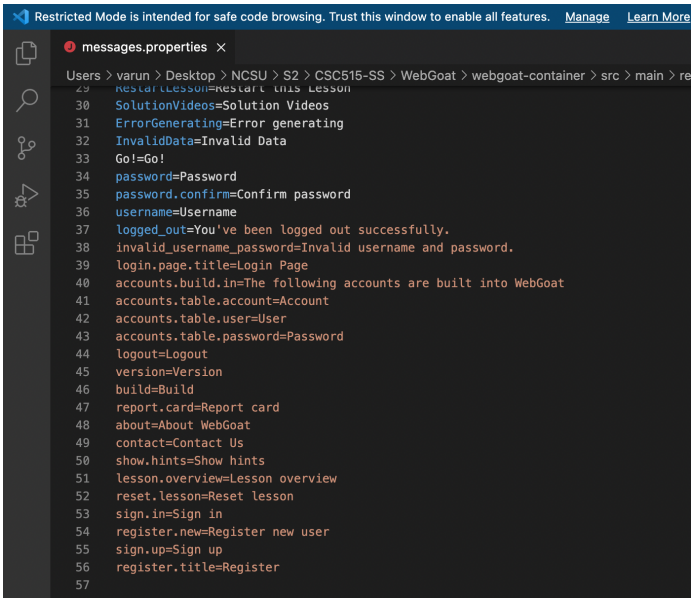
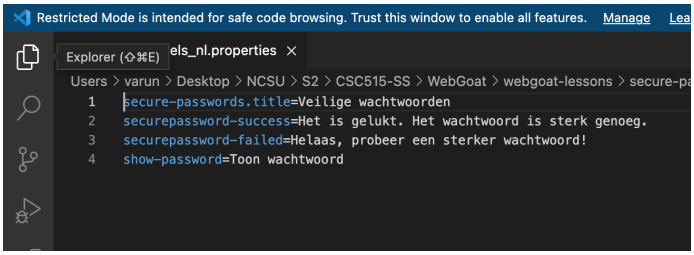
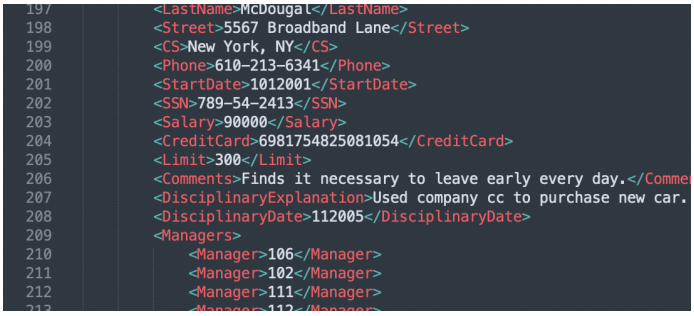
The files / directories each tool parses are different. Hence there is a possibility of having totally different results. For Ex: Github Security check only is going through the all metadata files like package.json to find the dependency issues. But OWASP is also parsing through the public folder to find other dependencies that are being imported into the given module.

Category	**Github's Checker**	**OWASP Dependency Checker**
Strength	This tool checks all the files like package-lock.json, composer.lock so the chances of finding the vulnerable dependencies are more	This tool actually tells whether the vulnerable dependency is transitive / direct
Weakness	This tool did not report any Vulnerabilities that are in "js" files that are located in public directory, this tool is essentially going through metadata json files like package-lock.json for any javascript dependencies and composer.lock.json for php related dependencies	This tool could only find vulnerable dependencies in package-lock.json and the actual js files present in the public directory like jquery and jquery-ui to find all the legacy versions in the given module
Usability	From a usability standpoint this is very easy to run on each commit / pull request because it is integrated into Github. By simply enabling the dependabot alerts, we can see the dependency vulnerabilities real time.	Since this is a third party tool and no inbuilt integration with the version control system, there is an extra effort needed to run on every incremental change in the repository.

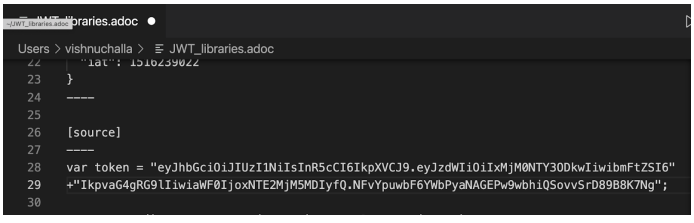
3. Secret Detection

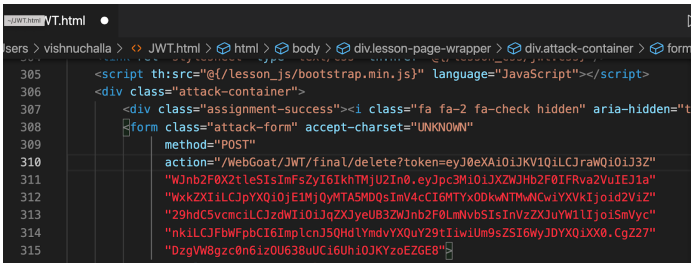
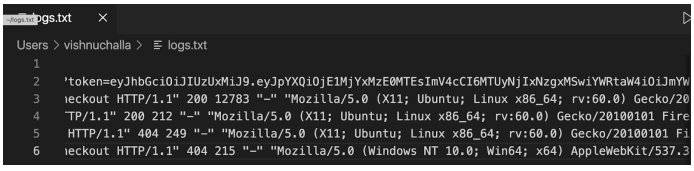
Webgoat - Whispers

Seq No	Secrets Detected	Screen Shot
1	Secret type: password Exposed Secret: Wachtwoord Line No: 31	
2	Secret type: accounts.table.password Exposed Secret: Wachtwoord Line No: 39	

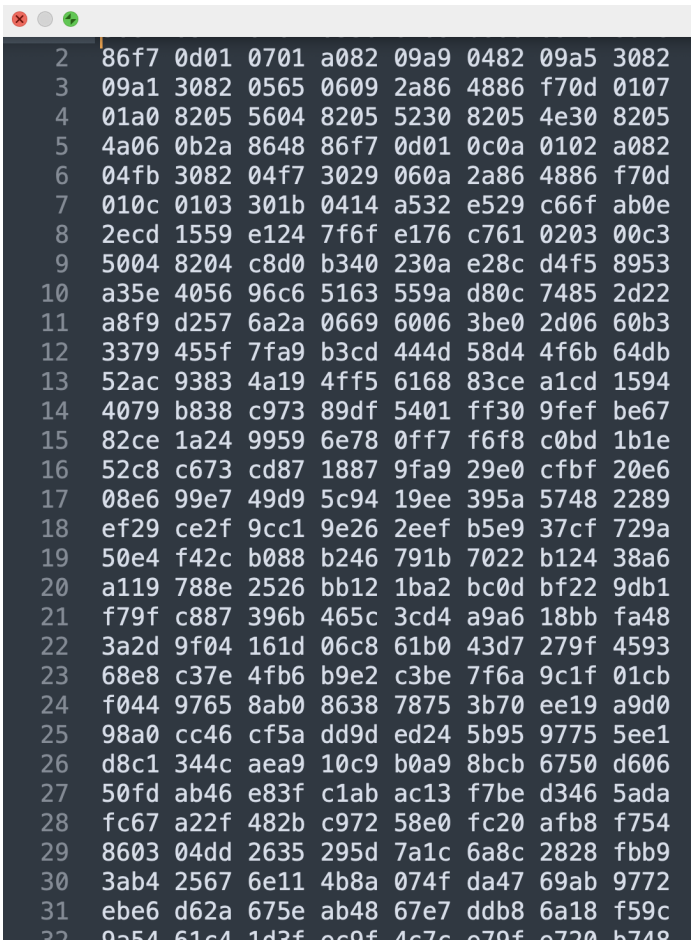
3	Secret type: accounts.table.password Exposed Secret: Password Line No: 43	
4	Secret type: show-password Exposed Secret: Toon wachtwoord Line No: 4	
5	Secret type: CreditCard Exposed secret: 6981754825081054 Line No: 204	

Webgoat - SpectralOps

Seq No	Secrets Detected	Screen Shot
1	Secret type: Token Exposed Secret: eyJraWQiOiJ3ZWJnb2F0X2tleSIsImFsZyI6IkhTNTEyIn0.eyJhdWQiOiJ3ZWJnb2F0Lm9yZyIsImVtYWlsIjoiamVycnlAd2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IkplcnJ5In0.xBc5FFwaOcuxjdr_VJ16n8Jb7vScuaZuINTI66F2MWF1aBe47QsUosvbjWGORNcMPiPNwnMu1Yb0WZVNrp2ZXA Line No: 27	
2	Secret type: Token Exposed Secret: eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1MjYxMzE0MTEsImV4cCI6MTUyNjIxNzgxMSwiYWRtaW4iOiJmYWxzZSIsInVzZXIiOiJUb20ifQ.DCoaq9zQkyDH25EcVWKcdbyVfUL4c9D4jRvsqOqvi9iAd4QuqmKcchfbU8FNzeBNF9tLeFXHZLU4yRkq-bjm7Q Line No: 76	
3	Secret type: Token Exposed Secret: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.NFvYpuwb	

	F6YWbPyaNAGEPw9wbhiQSovvSrD89B8K7Ng Line No: 28	
4	Secret type: Token Exposed Secret: eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE2MDgxMjg1NjYsImFkbWluIjoiZmFsc2UiLCJ1c2VyIjoiVG9tIn0.rTSX6PSXqUoGUvQQDBiqX0re2BSt7s2-X6FPf34Qly9SMpqIUSP8jykedJbjOBNlM3_CTjgk1SvUv48Pz8zIzA Line No: 35	
5	Secret type: action Exposed Secret: /WebGoat/JWT/final/delete?token=eyJ0eXAiOiJKV1QiLCJraWQiOiJ3ZWJnb2F0X2tleSIsImFsZyI6IkhTMjU2In0.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOjE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcmciLCJzdWIiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsInVzZXJuYW1lIjoiSmVycnkiLCJFbWFpbCI6ImplcnJ5QHdlYmdvYXQuY29tIiwiUm9sZSI6WyJDYXQiXX0.CgZ27DzgVW8gzc0n6izOU638uUCi6UhiOJKYzoEZGE8 Line No: 310	
6	Secret type: token Exposed Secret: eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1MjYxMzE0MTEsIm	

	V4cCI6MTUyNjIxNzgxMSwiYWRtaW4iOiJmYWxzZSIsInVzZXIiOiJUb20ifQ.DCoaq9zQkyDH25EcVWKcdbyVfUL4c9D4jRvsqOqvi9iAd4QuqmKcchfbU8FNzeBNF9tLeFXHZLU4yRkq-bjm7Q Line No: 2	
7	Secret type: token Exposed Secret: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0ZXN0IjoidGVzdCJ9.axNp9BkswwK_YRF2URJ5P1UejQNYZbK4qYcMnkusg6I Line No: 28	

8	Secret type: keystroke Exposed Secret: encrypted content attached in the screenshot. Line No: Entire file	

Common secrets:
We didn't find any common secrets among the three tools that we used. (Whispers, SpectralOps and GitLeaks)

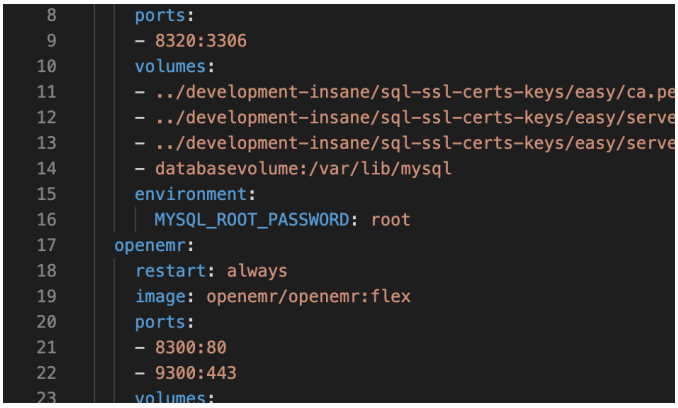
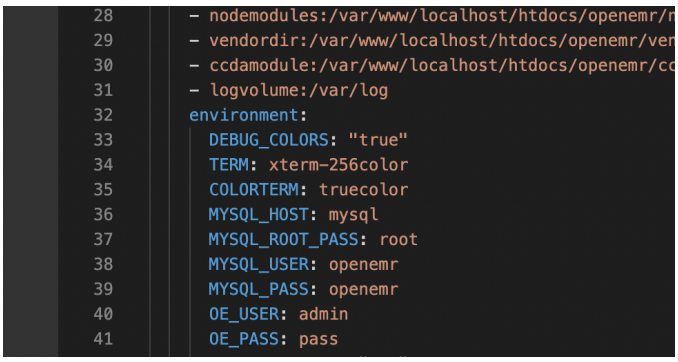
Why do tools differ?

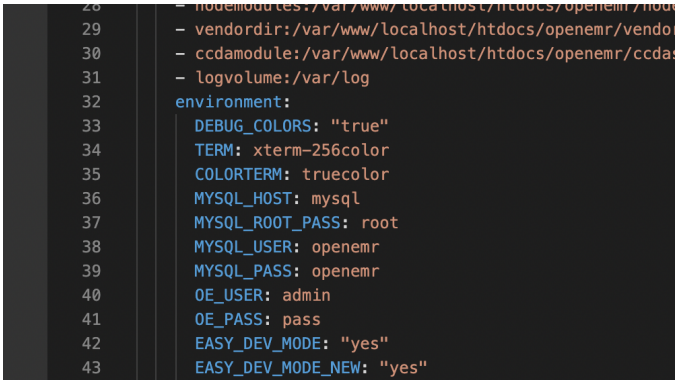
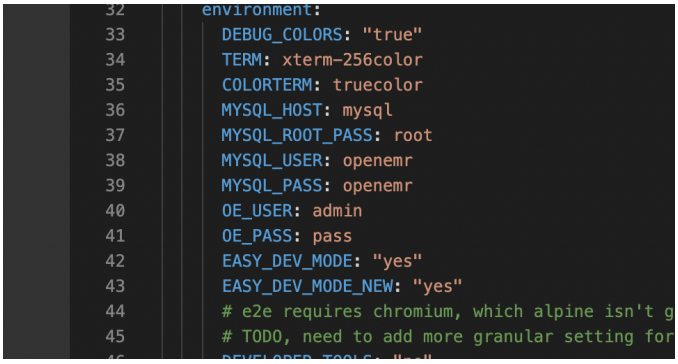
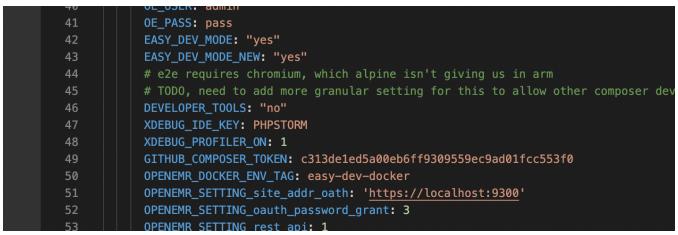
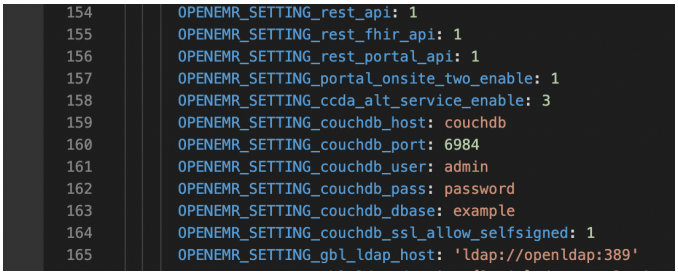
1. The keyword in search is different in different tools. For example, in the case of Whispers, it was able to find a credit card number but the other 2 tools didn't report that secret. This tells us that whispers have CreditCard as one of the keyword searches while searching for secrets but the other 2 didn't have that.
2. Spectral Ops use AI and Machine Learning to detect the secrets. So, it has to learn what is a secret before actually determining it. Hence the secrets caught by other tools were not reported by Spectral Ops.

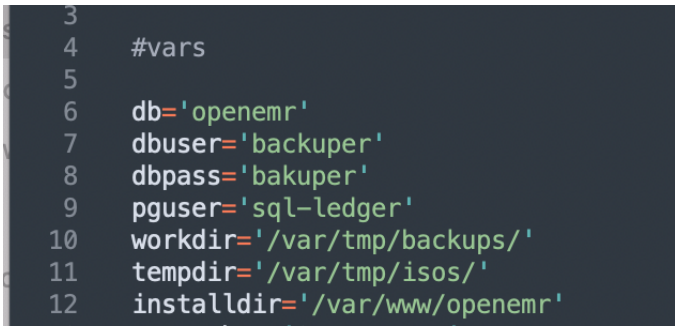
3. And when it comes to Whispers and GitLeaks, Whispers is implemented in python and GitLeaks in Go language. Due to cross platform implementations and differences in parse techniques used by the corresponding libraries, the conclusions of interpreting a secret may differ. This is what has resulted in zero match in our case.
4. And also the GitLeaks is undergoing active development and is adapting to the new secret detection techniques wherein Whispers last commit was in Oct 2021 which might be outdated according to today's secret detection technique. This might also be a reason for no matches between the tools.

Extra Credit

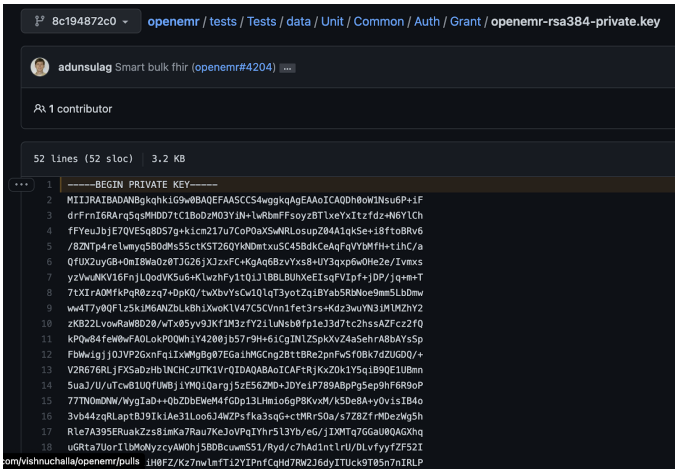
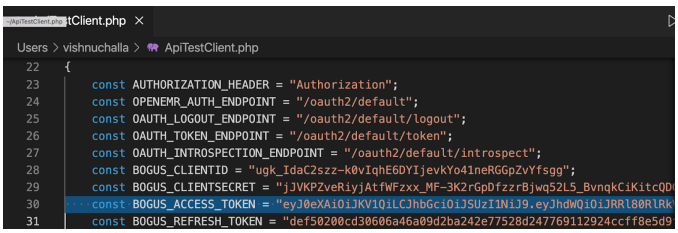
Openemr - Whispers

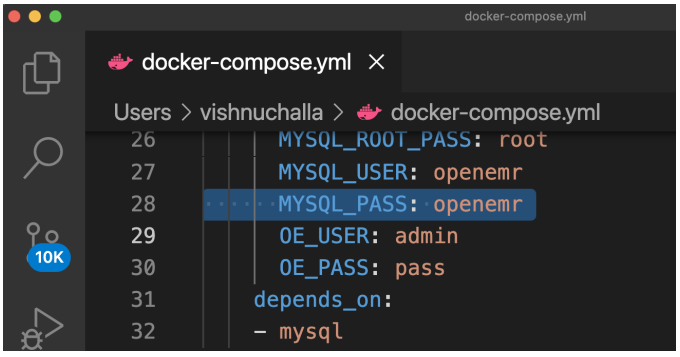
Seq No	Secrets	Screen Shots
1	Secret type: MYSQL_ROOT_PASSWORD Exposed Secret: root Line No: 16	
2	Secret type: MYSQL_ROOT_PASS Exposed Secret: root Line No: 37	

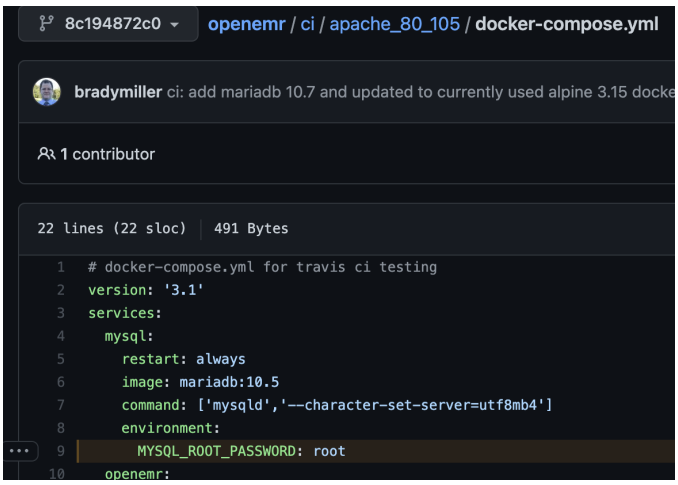
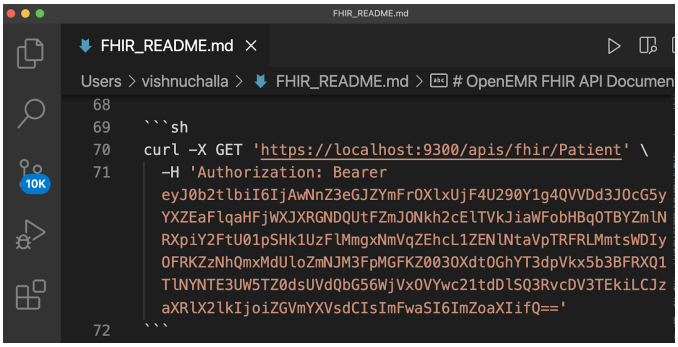
3	Secret type: MYSQL_PASS Exposed Secret: openemr Line No: 39	
4	Secret type: OE_PASS Exposed Secret: pass Line No: 41	
5	Secret type: GITHUB_COMPOSER_TOKEN Exposed Secret: c313de1ed5a00eb6ff9309559ec9ad01fcc553f0 Line No: 49	
6	Secret type: OPENEMR_SETTING_couchdb_pass Exposed Secret: password Line No: 162	

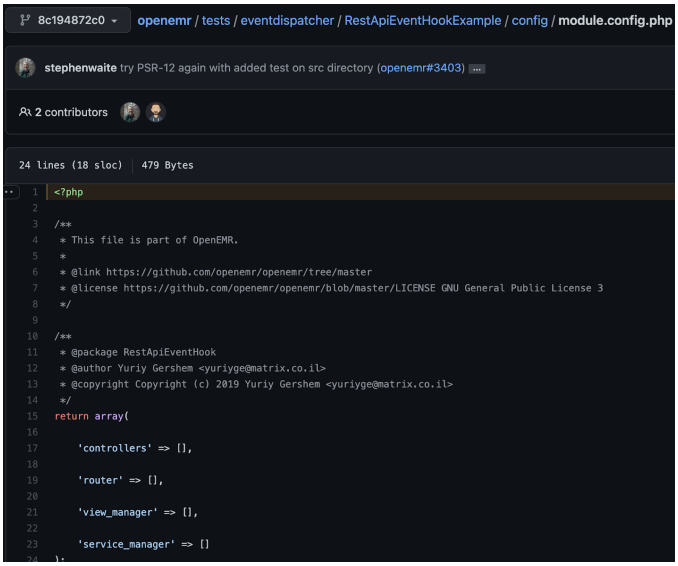
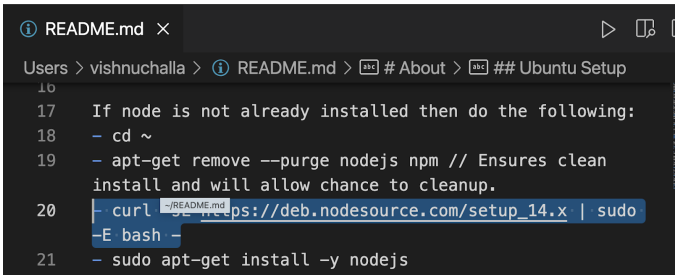
7	Secret type: dbpass Exposed Secret: bakuper Line No: 8	

Openemr - SpectralOps

Seq No	Secrets	Screen Shots
1	Secret type: private key Exposed Secret: encrypted content attached in the screenshot. Line No: Entire file	
2	Secret type: token Exposed Secret: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiJRRl80RlRkV2h4eGJlb1Y4SGU5c1ZRQUl3STlQZWdYM3IyVUdiTTVjaWtNIiwianRpIjoiYWZmYWEyOTk5NWY5MjRjZDlmMjc0YTdhZGM5NmM0YzJmZTYwNGE3MjA0ODFkMTdmMWZiYTg2ZDg4ZWIxO	

	WI0YWVlM2RhZTM1Zjg3MjcwMDAiLCJpYXQiOjE2MDcyMTc5MTksIm5iZiI6MTYwNzIxNzkxOSwiZXhwIjoxNjA3MjIxNTE5LCJzdWIiOiI5MjJjYzYzNi01NTNiLTQ1MGQtYTJkZC1hNmRjYmM4ZDNiMDciLCJzY29wZXMiOlsib3BlbmlkIiwic2l0ZTpkZWZhdWx0Il19.u2Ujtln3vEKIR8E0rSd-xr6m-3huCxiFMaTm9_NQplVNsBkrc6Y3KLy9FRZVS3xSY1Qgav-UvOxikYT2zNNlK-LotEoEvZdtj87X6fh4wh-h5BU87lHh9aNjXFUemSO9DGKJLSZdLSeC_2w4YmbhQykGFISiltD2_PAJRKuKbpcBo3-Lafe2N83mF5i8mZXSCu_fbLrmTMYPCnsRaU_sWStzFp6p0SM3zGfLt1kjw-hcE82Ci1puqRS2nR5Z3kEOXz-hQOdXmQMq0s_gkeQZvLPOJwLGfEX5d4eIU4BfngksjGkKQhC7rUKT-_2F-U_z30P3izzZM6m4dZ10liP80g Line No: 30	
3	Secret type: password Exposed Secret: openemer Line No: 28	

4	Secret type: password Exposed Secret: root Line No: 9	
5	Secret type: Authorization token Exposed Secret: Bearer eyJ0b2tlbiI6IjAwNnZ3eGJZYmFrOXlxUjF4U290Y1g4QVVDd3JOcG5yYXZEaFlqaHFjWXJXRGNDQUtFZmJONkh2cElTVkJiaWFobHBqOTBYZmlNRXpiY2FtU01pSHk1UzFlMmgxNmVqZEhcL1ZENlNtaVpTRFRLMmtsWDIyOFRKZzNhQmxMdUloZmNJM3FpMGFKZ003OXdtOGhYT3dpVkx5b3BFRXQ1TlNYNTE3UW5TZ0dsUVdQbG56WjVxOVYwc21tdDlSQ3RvcDV3TEkiLCJzaXRlX2lkIjoiZGVmYXVsdCIsImFwaSI6ImZoaXIifQ== Line No: 71	

6	Secret type: php tag Exposed Secret: “<?php” This is a false positive but was detected by the tool Line No: 1	
7	Secret type: URL Exposed Secret: curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash - This is also a false positive but was detected by the tool Line No: 20	

What openemr should do to protect the secrets you found?

1. All the secrets should not be stored within the code. Instead, it should be stored in a vault within the cloud where the application is deployed. The application should fetch from the vault where it’s deployed.
2. Authorization tokens should neither be stored in code nor it should be cached. Instead, it should be fetched from network calls, protected by SSL, TLS protocol. For eg: OAuth.

Do these secrets appear to protect valuable resources/assets? Provide the valuable assets information protected by the secrets

1. DB secrets like MYSQL_ROOT_PASS, MYSQL_USER, MYSQL_HOST are exposed. Database tables, data within each table can be accessed using these resources.
2. Authorization token to retrieve patient information is exposed. The GET end-point combined with the authorization token can be used to retrieve patients' sensitive information like health records, social security number, phone number, license id.
3. The private key is visible in the code. We can use this private key to decrypt any encrypted content found across openemr applications which might have been encrypted using the public key generated by some asymmetric encryption technique.

What good design principles does it seem OpenEMR has done to protect other types of secrets?

1. Cryptography - To maintain the Confidentiality, Authenticity, and Integrity of the data, the openemr application has used some asymmetric encryption techniques to generate a public-private key pair to encrypt and decrypt the data. One of them we found was the PGP encryption technique being used which also verifies digital signature while decrypting the data.