Security Assessment

Lenez Adam

Assignment Overview

As part of this assignment, the focus is on assessing the security posture of a colleague's application, emphasizing sound secure development practices. The assigned roles involve the application owner providing access to both the runtime and codebase by November 12th, enabling the assessor to conduct a comprehensive evaluation. The assessor's primary responsibility is to evaluate the application using Static Analysis Security Testing (SAST), Dynamic Analysis Security Testing (DAST), and Software Composition Analysis (SCA). The emphasis is on ensuring secure development practices rather than penetration testing.

Introduction to the Application

Assessed

`Youssef Boudiba`

Application - Mind Vault

https://a8462f3ftrial.launchpad.cfapps.us10.hana.ondemand.com/site?siteId=137c66b4-7048-421f-b0fc-2645bd06c0d0#Shell-home

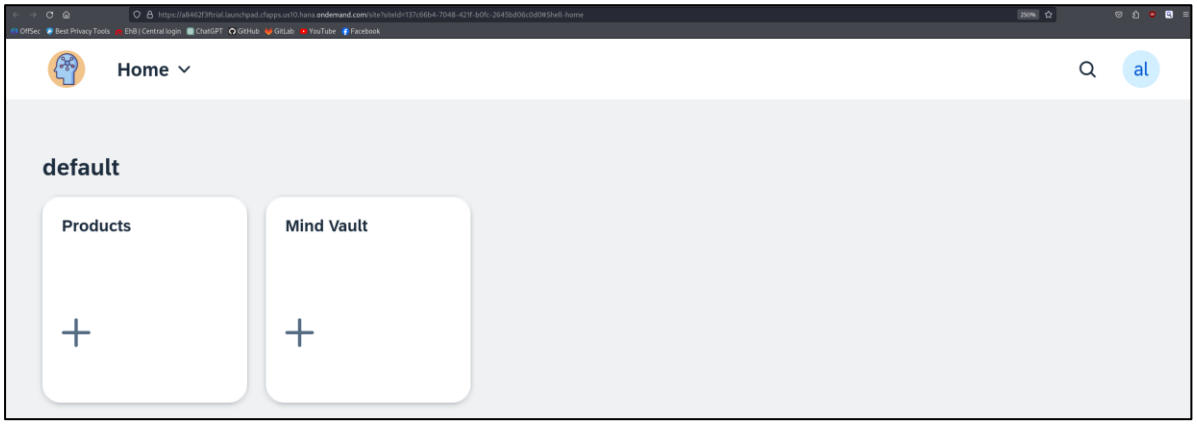
GitLab repository

https://gitlab.com/trial5180545/open-mind-vault

Application Overview

The application requires you to log in with a SAP-user account. Following tutorial was given to me to create the appropriate account: https://developers.sap.com/tutorials/hcp-create-trial-account.html
The account email address was requested by Yousef for authentication.

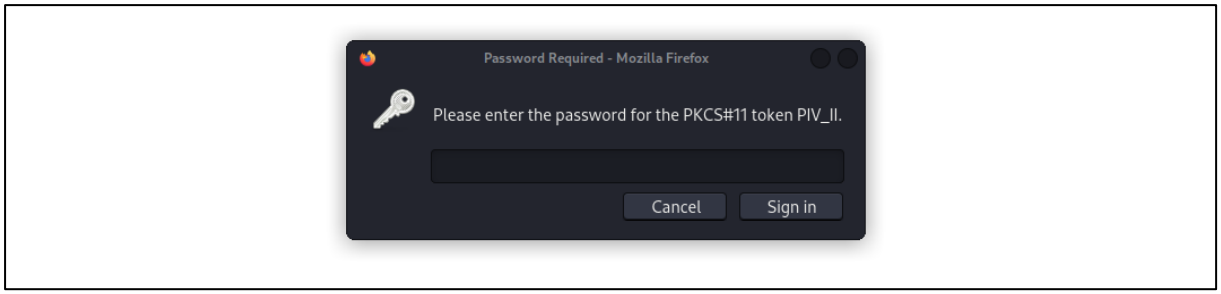
When logging in I was presented with following view:



I have asked Youssef what the products tab is about, and he told me to ignore that part since it only existed for testing purposes.

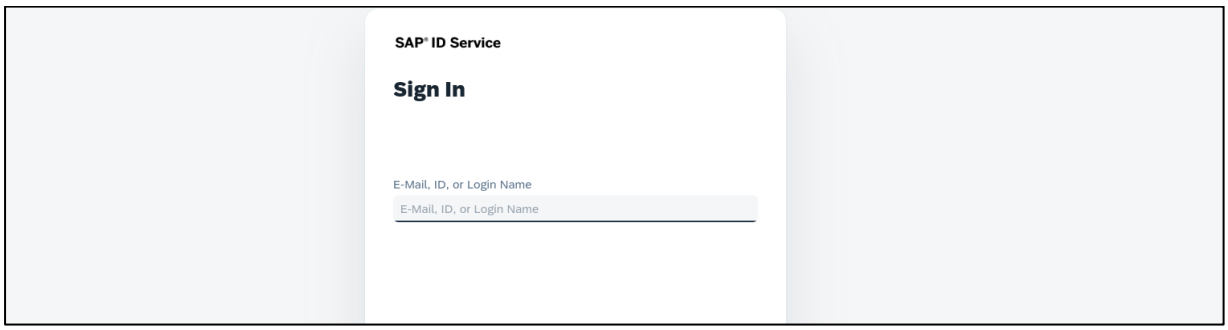
As for the Mind Vault, users can use it as some sort of Twitter application where people can post their thoughts publicly. Youssef has mentioned a bug that does not automatically refreshes the page when posting a thought. I tried this and indeed, no response when posting my thought. Tried it three times before refreshing with following as the result. (image below)

I have noticed an option to post your thoughts publicly. I was wondering if these public posts required an account to view, so I opened an incognito tab and browsed to the Mind Vault link. I was welcomed by a login pop-up.

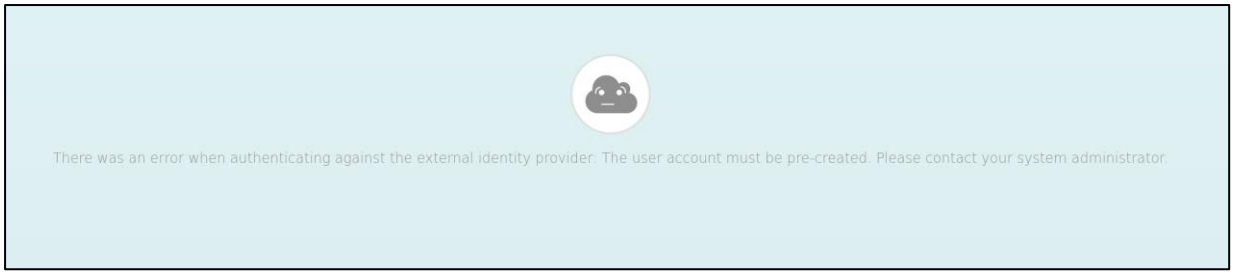


This window required me to enter my password. It was a persistent window where I could only exit the process when I repeatedly cancelled or exited the browser window. Pretty unpleasant user experience since it stopped all my browsing processes across Firefox, this meant other tabs and windows where I was working as well. For example, I was not able to work in my online word processor until I entered the password, or repeatedly cancelled the pop up. At first glance it seems Youssef has built the application with security as top priority.

You may have noticed the window did not ask me for a username or email address. The option is available when cancelling the pop-up multiple times. Eventually you will get a Sign In form seen in the image below.



Here you could sign in with an alternative SAP user account. I created a second account, without forwarding the account to Youssef and was presented with an authentication error as expected when logging in.



Summary of Findings:

Authentication Process:

- The application relies on SAP user accounts for authentication.
- Account creation tutorial: SAP User Account Creation Tutorial.
- Email address submission to Youssef for authentication.

User Interface and Functionality:

- The Products tab is explained as existing for testing purposes and can be ignored.
- Mind Vault serves as a public thought-sharing platform, resembling a Twitter application.
- Identified bug: Automatic page refresh upon posting a thought is not functioning.
- Attempted posting resulted in no response until a manual page refresh.

Public Post Accessibility:

- Explored whether public posts require an account for viewing.
- Opened Mind Vault link in an incognito tab, encountered a login pop-up.
- The pop-up window persisted and required a password, causing inconvenience and disrupting other browser activities.
- Initial impression of a security-focused application.

Login Pop-up Behaviour:

- Pop-up did not prompt for a username or email address initially.
- Cancelling the pop-up multiple times revealed the option to input a username or email address.

User Experience Issues:

- Persistent login pop-up causing disruptions and inconvenience in other browser activities.

Alternative SAP User Account Handling:

- Created a second SAP user account without forwarding it to Youssef.
- Authentication error encountered as expected, indicating proper handling of unauthorized access attempts.

Scope and Methodology

The assessment will focus on evaluating the security posture of Yousef's application, with a specific emphasis on the following components:

User Authentication Mechanism:

- Verification of secure user authentication practices.
- Examination of the process for creating SAP user accounts as guided by the provided tutorial.

Access Control Policies:

- Assessment of how access control is enforced within the application.
- Evaluation of user roles and permissions to ensure appropriate access levels.

Communication Channels:

- Inspection of the security measures implemented for communication between different components of the system.
- Verification of data protection during inter-component communication.

Web Vulnerabilities and Countermeasures:

- Identification and analysis of potential web vulnerabilities within the application.
- Assessment of the effectiveness of countermeasures in place to mitigate these vulnerabilities.

User Privacy Protection:

- Evaluation of measures in place to protect the privacy of users, especially concerning public posts in the Mind Vault.

Application Deployment and Operations Code:

- Analysis of code responsible for deploying the application, including any automated processes.
- Inspection of operational code to ensure security measures are integrated.

Methodologies Employed:

Static Analysis Security Testing (SAST):

- **Tool:** Semgrep
- **Explanation:** SAST involves a thorough examination of the application's source code and related artifacts without executing the program. Semgrep, being a versatile static analysis tool, will be employed to identify potential security vulnerabilities, coding flaws, and adherence to secure coding practices.

Dynamic Analysis Security Testing (DAST):

- **Tool:** OWASP Zed Attack Proxy (ZAP)
- **Explanation:** DAST involves analyzing the application while it's running to identify vulnerabilities that may be exploited in real-world scenarios. ZAP, a well-known and widely used DAST tool, will be utilized to simulate potential attacks, assess runtime security, and identify vulnerabilities that might not be apparent through static analysis alone.

Software Composition Analysis (SCA):

- **Tool:** OWASP Dependency-Check
- **Explanation:** SCA is crucial for identifying and managing vulnerabilities within third-party dependencies. OWASP Dependency-Check will be employed to scrutinize the application's external components, ensuring that known vulnerabilities are identified, and appropriate actions are taken to address them.

Note: The choice of tools may be subject to change during the exploration phase, depending on the specific requirements and findings. Any adjustments made will be clearly documented in the final assessment report. The combination of SAST, DAST, and SCA methodologies ensures a comprehensive evaluation of the application's security from both static and dynamic perspectives, as well as a thorough examination of its external dependencies.

The assessment will be structured into three distinct phases, each focusing on a specific security testing methodology: Static Analysis Security Testing (SAST), Dynamic Analysis Security Testing (DAST), and Software Composition Analysis (SCA).

1. SAST Assessment
    - **Objective:** Identify code-level vulnerabilities without executing the application.
    - **Tool:** Semgrep.
    - **Scope:** User authentication, access control, communication channels, web vulnerabilities, and privacy protection.
    - **Structure:** Introduction, setup details, methodology, findings, recommendations, conclusion.
2. DAST Assessment
    - **Objective:** Evaluate runtime security through simulated real-world attacks.
    - **Tool:** OWASP ZAP.
    - **Scope:** Same as SAST.
    - **Structure:** Introduction, setup details, methodology, findings, recommendations, conclusion.
3. SCA Assessment
    - **Objective:** Analyze and manage security and compliance issues in third-party components.
    - **Tool:** OWASP Dependency-Check.
    - **Scope:** Examination of external dependencies.
    - **Structure:** Introduction, setup details, methodology, findings, recommendations, conclusion.

This structured approach should ensure a comprehensive evaluation of the application's security, covering code-level vulnerabilities, runtime security, and third-party component risks. Each phase will contribute valuable insights to the overall security posture.

SAST Assessment

Introduction to SAST Assessment

Static Analysis Security Testing (SAST) is a crucial security testing methodology that focuses on evaluating the security of an application by examining its source code, configuration files, and binaries without executing the program. Unlike dynamic testing methods, SAST is performed during the development phase and can identify potential security vulnerabilities early in the software development lifecycle.

In summary, Static Analysis Security Testing is a proactive approach to identifying and addressing security vulnerabilities in the early stages of software development. It plays a crucial role in building secure, high-quality applications by enabling developers to identify and mitigate potential security risks before deployment.

Access and Environment Setup for SAST

Accessing the Application's Source Code:

Youssef has provided me with a link and gave me access to his GitLab repository.

https://gitlab.com/trial5180545/open-mind-vault

With help from Youssef, I was able to deploy and run the application locally. I had to complete following steps.

1. Create files `default-env.json` and `default-services.json` with data received from Youssef.
2. Paste both files in the root and dev-approuter folders
3. Run `npm install` where `package.json` files are located
4. Run `npm run start` in `./dev-approuter` folder
5. Run `npm run start` in rootfolder

Setting Up Semgrep:

Semgrep should be installed on the assessment environment. This can be achieved through package managers like pip or by following the installation instructions provided by the Semgrep documentation (https://semgrep.dev/docs/semgrep-code/getting-started/).

```
1. python3 -m pip install semgrep
2. export PATH=$PATH:$(python3 -m site --user-base)/bin
3. semgrep login
4. semgrep ci
```

SAST Methodology

This chapter provides insights into the Static Analysis Security Testing (SAST) methodology applied using Semgrep. It encompasses an explanation of how Semgrep, a powerful static analysis tool, was employed for code analysis. The overview outlines the techniques used, including pattern matching, AST analysis, flow-sensitive analysis, and context-aware matching.

Pattern Matching with Code Patterns

Semgrep uses code patterns, written in the Semgrep pattern language, to identify specific sequences of code that match known security vulnerabilities or coding patterns.

How it Works with Semgrep:

Users can define custom patterns or use existing patterns from the Semgrep rules registry.

Patterns can be written using a combination of syntax, data flow, control flow, and other context-aware constructs.

AST (Abstract Syntax Tree) Matching

Semgrep analyzes the abstract syntax tree of the code to understand its structure and relationships between different elements.

How it Works with Semgrep:

Semgrep patterns can be written to match specific AST structures, allowing for fine-grained control over the analysis.

This approach helps in identifying vulnerabilities based on the hierarchical structure of the code.

Flow-Sensitive Analysis

Semgrep performs flow-sensitive analysis to understand how data flows through the application and identify security-sensitive operations.

How it Works with Semgrep:

Patterns in Semgrep can be written to track the flow of data, allowing the tool to catch issues related to how data is processed or manipulated across different parts of the code.

Context-Aware Matching

Semgrep is context-aware, allowing patterns to be matched based on the surrounding context and code constructs.

How it Works with Semgrep:

Users can define patterns that take into account the context in which certain code constructs appear, enhancing the accuracy of the analysis.

Language-Agnostic Analysis

Semgrep supports multiple programming languages, making it language-agnostic and versatile for analyzing codebases written in different languages.

How it Works with Semgrep:

Semgrep can be used for analyzing code written in languages like Python, JavaScript, Java, Go, and more, making it a flexible choice for diverse codebases.

Regular Expression Support

Semgrep supports regular expressions within patterns, allowing for more flexible and powerful matching capabilities.

How it Works with Semgrep:

Regular expressions can be incorporated into Semgrep patterns to capture complex code patterns or variations.

SAST Findings

After the scan was completed, I was able to look up its findings. To my surprise, it only had two open findings. Both findings were about missing integrity. A medium severity vulnerability with low confidence.

The `integrity` attribute allows for the browser to verify that externally hosted files (for example from a CDN) are delivered without unexpected manipulation. Without this attribute, if an attacker can modify the externally hosted resource, this could lead to XSS and other types of attacks. To prevent this, include the base64-encoded cryptographic hash of the resource (file) you're telling the browser to fetch in the `integrity` attribute for all externally hosted files.

https://semgrep.dev/r?q=html.security.audit.missing-integrity

Here is an example of the integrity rule. This made clear that this was indeed a correctly flagged vulnerability. The source path does seem to be reaching out without using an integrity tag.

Recommendations and Remediation for SAST Findings:

After the SAST scan using Semgrep, two open findings were identified, both related to missing integrity, presenting a medium-severity vulnerability with low confidence. The vulnerabilities center around the absence of the integrity attribute in externally hosted files, posing potential risks such as XSS attacks if the files are manipulated by attackers.

Actionable Recommendations

Integrate Integrity Tags

Implement integrity tags for all externally hosted files to enable the browser to verify the integrity of resources, especially those from CDNs. This involves including the base64-encoded cryptographic hash of the resource in the integrity attribute.

Guidance on Severity and Urgency

Severity Level: Medium

Reasoning: While the severity is medium, the impact can be significant, as it relates to the integrity of externally hosted files. This could lead to XSS and other attacks if not addressed.

Urgency for Remediation: Address this issue in a timely manner to prevent potential exploitation. Though the confidence level is low, the impact on integrity is critical.

Additional Recommendations Based on OWASP Information (A08:2021)

Digital Signatures

Utilize digital signatures or similar mechanisms to verify that software or data is from the expected source and has not been altered. Using the integrity tag would satisfy this requirement.

Secure Libraries and Dependencies

Ensure that libraries and dependencies are sourced from trusted repositories. Consider hosting an internal repository for higher-risk profiles.

Software Supply Chain Security

Implement a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, to verify components for known vulnerabilities.

Code and Configuration Review

Establish a thorough review process for code and configuration changes to minimize the introduction of malicious code or configuration into the software pipeline. Looking at the GitLab repository I see that there is no pipeline configured.

Secure CI/CD Pipeline

Ensure proper segregation, configuration, and access control within the CI/CD pipeline to maintain the integrity of code flowing through build and deploy processes. Again, no pipeline is configured with this project.

Encrypted Serialized Data

Avoid sending unsigned or unencrypted serialized data to untrusted clients without an integrity check or digital signature to detect tampering or replay of the serialized data.

Addressing these recommendations comprehensively will not only remediate the identified vulnerabilities but also enhance the overall software and data integrity of the application.

Conclusion for SAST Assessment:

Discovering just two medium-severity vulnerabilities through the SAST assessment using Semgrep was a surprising experience. These findings shed light on the importance of seemingly subtle aspects, specifically the absence of integrity tags in externally hosted files. While the confidence level was initially low, the potential impact on integrity and the looming threat of XSS attacks underscored the need for immediate attention.

Reflecting on the overall security posture, it's evident that even minor vulnerabilities demand careful consideration. Strengthening the application's resilience requires a proactive approach to address potential exploits, no matter their perceived severity.

DAST Assessment

Introduction to DAST Assessment

Dynamic Analysis Security Testing (DAST) stands as a pivotal methodology in evaluating the runtime security of an application. Unlike its static counterpart (SAST), DAST focuses on assessing the application's security posture during execution. This dynamic approach simulates real-world attack scenarios, providing insights into vulnerabilities and weaknesses that might be exploited by malicious actors when the application is live.

DAST operates by actively probing the running application, mimicking how an external attacker would interact with it. This process uncovers vulnerabilities that might not be apparent in the source code alone, emphasizing the significance of evaluating security in a real-world runtime environment. Through its dynamic nature, DAST captures vulnerabilities related to input validation, session management, authentication mechanisms, and other runtime-specific issues.

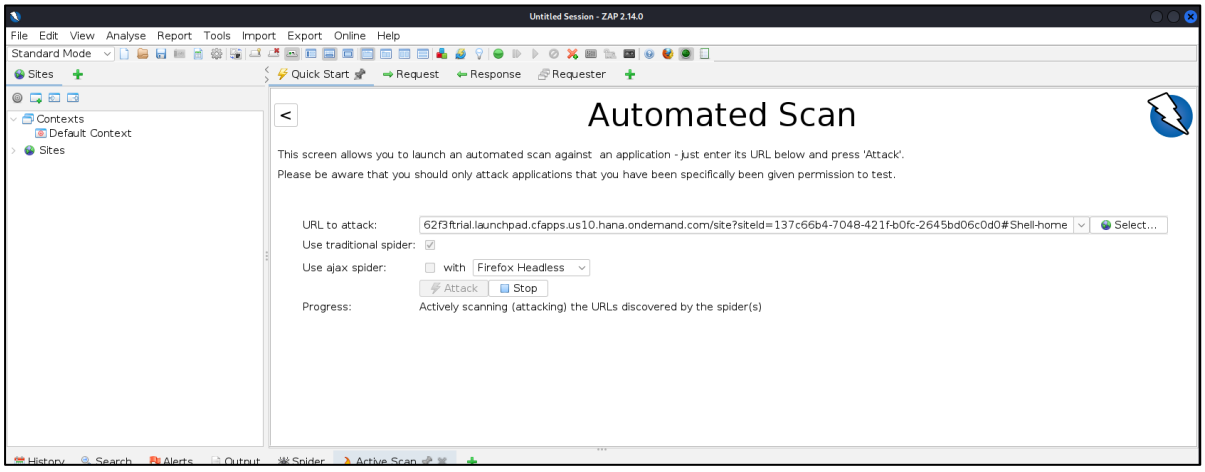
In this assessment, the goal is to scrutinize the application's behavior under dynamic conditions, identifying potential entry points for exploitation and ensuring robust defenses against common attack vectors. DAST plays a crucial role in providing a holistic view of the application's security, complementing static analysis efforts and contributing to a comprehensive security strategy.

Access and Environment Setup for DAST

Previous experience has shown that the ZAP tool can act buggy when used on windows. So for this assignment I have dual booted my machine with Kali Linux. The ZAP tool was not preinstalled on the distro but with one simple command I got it working.

```
sudo apt install zaproxy
```

After launching and updating the ZAP library packages, I browsed to the link given by Youssef and launched an automated scan, using traditional spider.



DAST Methodology

In this DAST methodology, OWASP ZAP played a central role in simulating attacks and evaluating the runtime security of the application. ZAP, being a dynamic analysis tool, actively interacts with the live application, emulating the behaviour of potential attackers to identify vulnerabilities and security weaknesses.

Dynamic Analysis Techniques

Spidering

Description: ZAP utilizes spidering to comprehensively map the application's structure, identifying all accessible pages, endpoints, and parameters.

Purpose: Spidering aids in establishing a complete understanding of the application's surface, ensuring that all areas are subject to subsequent dynamic analysis.

Active Scanning

Description: ZAP performs active scanning by sending crafted requests to various parts of the application, actively probing for vulnerabilities.

Purpose: Active scanning allows ZAP to identify and exploit potential security issues, such as injection flaws, cross-site scripting (XSS), and other runtime-specific vulnerabilities.

AJAX Spidering

Description: ZAP incorporates AJAX spidering to handle dynamic content and interactions within the application, ensuring that modern web applications with asynchronous behavior are thoroughly assessed.

Purpose: AJAX spidering enables ZAP to navigate through pages and components that rely on asynchronous requests, capturing potential vulnerabilities in dynamically loaded content.

Fuzzer

Description: ZAP employs a fuzzer to systematically inject malicious input into parameters, headers, and other user-controllable inputs, seeking to uncover vulnerabilities related to input validation.

Purpose: The fuzzer helps identify potential injection vulnerabilities, parameter tampering, and other issues that may arise when untrusted data is processed.

Authentication Testing

Description: ZAP includes features for testing authentication mechanisms, allowing the tool to handle login sessions and navigate through authenticated areas of the application.

Purpose: Authentication testing ensures that ZAP can effectively analyze secured sections of the application, mimicking user interactions under various access levels.

Session Management Analysis

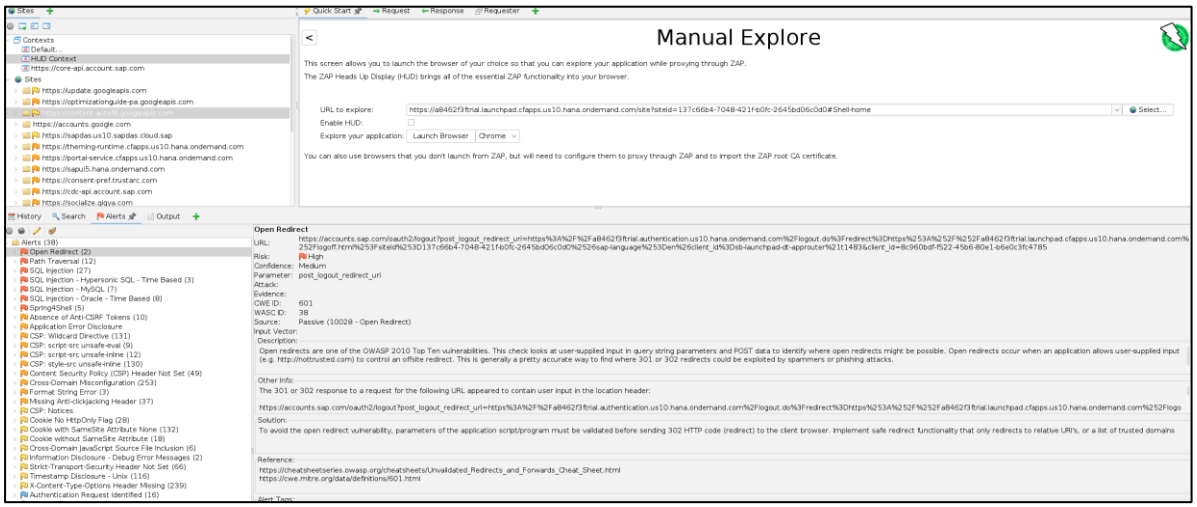
Description: ZAP assesses the application's session management to identify potential weaknesses related to session tokens, cookies, and related security controls.

Purpose: Analyzing session management helps uncover vulnerabilities that may lead to unauthorized access or session hijacking.

By employing these dynamic analysis techniques, ZAP provides a robust and comprehensive assessment of the application's runtime security. The combination of active scanning, spidering, AJAX handling, and targeted testing methodologies enhances the tool's effectiveness in identifying and mitigating security risks in real-world scenarios.

DAST Findings

I have scanned the application link with an automated and manual scan using traditional and ajax spider as well did the active scan. I have done all types of scans under the default context and HUD context, where I was able to authenticate the user and spider the URLs that would have been unauthorized otherwise.



After scanning the complete application, ZAP had found 38 alerts. I was able to generate a report about the findings. All alerts are very well described in the document. I will write a high-level overview of the document but for more information, please open file ZAP Scanning Report.

Alert Counts by Risk and Confidence:

- **High Risk, Medium Confidence:** 7 alerts (18.4%)
- **High Risk, Low Confidence:** 1 alert (2.6%)
- **Medium Risk, High Confidence:** 10 alerts (26.3%)
- **Medium Risk, Medium Confidence:** 9 alerts (23.7%)
- **Medium Risk, Low Confidence:** 1 alert (2.6%)
- **Low Risk, High Confidence:** 9 alerts (23.7%)
- **Low Risk, Medium Confidence:** 6 alerts (15.8%)
- **Low Risk, Low Confidence:** 1 alert (2.6%)
- **Informational Risk, Medium Confidence:** 6 alerts (15.8%)
- **Informational Risk, Low Confidence:** 6 alerts (15.8%)

Alert Counts by Alert Type:

- **Open Redirect:** 2 alerts (5.3%)
- **Path Traversal:** 12 alerts (31.6%)
- **SQL Injection:** 27 alerts (71.1%)
- **SQL Injection - Hypersonic SQL - Time Based:** 3 alerts (7.9%)
- **SQL Injection - MySQL:** 7 alerts (18.4%)
- **SQL Injection - Oracle - Time Based:** 8 alerts (21.1%)
- **Spring4Shell:** 5 alerts (13.2%)
- **Absence of Anti-CSRF Tokens:** 10 alerts (26.3%)
- ... (Other alert types are also listed)

Key Findings:

- SQL Injection seems to be a significant concern, with various instances identified.
- Path Traversal and Open Redirect vulnerabilities are also present.
- Content Security Policy (CSP) issues, including missing headers and unsafe directives, are reported.
- Cross-Domain Misconfiguration and Cross-Site Request Forgery (CSRF) vulnerabilities are mentioned.
- Information Disclosure and Cookie-related issues have been identified.

Recommendations:

- **Urgent Fixes:** Prioritize fixing high-risk vulnerabilities with medium to high confidence.
- **SQL Injection Mitigation:** Implement proper input validation and parameterized queries to prevent SQL injection.
- **CSP Implementation:** Ensure proper Content Security Policy headers are set to mitigate content injection attacks.
- **Path Traversal Prevention:** Validate and sanitize user inputs to prevent path traversal vulnerabilities.
- **Secure Cookie Practices:** Review and enhance cookie attributes to ensure secure handling.

This is a high-level overview, refer to the detailed report for specific URLs, parameters, and recommendations associated with each finding.

Recommendations and Remediation for DAST Findings

Since the application is deployed in an SAP environment, I will focus on the core of the application and authentication URL's and not so much on the other services and API's qthat SAP is calling to. This would include following URL's:

https://core-api.account.sap.com
https://a8462f3ftrial.authentication.us10.hana.ondemand.com
https://a8462f3ftrial.launchpad.cfapps.us10.hana.ondemand.com

For more information about the remediation, you can always read the detailed report where you will be pointed to official links about found vulnerabilities remediations.

1. Cross-Domain Misconfiguration (Medium Risk, Medium Confidence)
https://core-api.account.sap.com

Description: The CORS misconfiguration on the web server allows cross-domain read requests from arbitrary third-party domains, exposing potential security risks. Although authenticated APIs are somewhat protected, the misconfiguration could still be exploited to access unauthenticated data.

Recommendation:

- **Ensure Sensitive Data Protection:** Confirm that sensitive data is not accessible in an unauthenticated manner, such as through IP address white-listing.
- **Review CORS Headers:** Configure the "Access-Control-Allow-Origin" HTTP header more restrictively or consider removing all CORS headers to enforce the Same Origin Policy (SOP) more strictly.

2. Cookie with SameSite Attribute None (Low Risk, Medium Confidence)
https://a8462f3ftrial.authentication.us10.hana.ondemand.com

Description: A cookie has been set with the SameSite attribute set to "none," allowing it to be sent as a result of a 'cross-site' request. This poses a potential security risk.

Recommendation:

- **Enhance SameSite Attribute:** Set the SameSite attribute to either 'lax' or ideally 'strict' for all cookies to improve security.

3. Cross-Domain JavaScript Source File Inclusion (Low Risk, Medium Confidence)
https://a8462f3ftrial.launchpad.cfapps.us10.hana.ondemand.com

Description: The page includes one or more script files from a third-party domain, which can introduce security vulnerabilities.

Recommendation:

- **Review Third-Party Scripts:** Carefully assess and validate the necessity of third-party scripts. Consider hosting essential scripts locally or from trusted sources.

4. Content Security Policy (CSP) Header Not Set (High Risk, Medium Confidence)
https://a8462f3ftrial.authentication.us10.hana.ondemand.com

Description: Content Security Policy (CSP) header is not set, leaving the application vulnerable to various types of attacks, including Cross-Site Scripting (XSS) and data injection attacks.

Recommendation:

- **Implement CSP Header:** Ensure that your web server, application server, load balancer, etc., are configured to set the Content-Security-Policy header.

5. CSP: Wildcard Directive (Medium Risk, Medium Confidence)
https://a8462f3ftrial.launchpad.cfapps.us10.hana.ondemand.com

Description: The CSP includes wildcard directives for certain sources, potentially allowing a broad range of sources that may introduce security risks.

Recommendation:

- **Refine CSP Directives:** Review and refine the CSP directives, specifically addressing wildcard usage for sources like style-src, img-src, connect-src, etc., to minimize potential vulnerabilities.

6. CSP: script-src unsafe-inline (Medium Risk, Medium Confidence)
https://a8462f3ftrial.launchpad.cfapps.us10.hana.ondemand.com

Description: The CSP includes 'unsafe-inline' for script-src, which can expose the application to script injection attacks.

Recommendation:

- **Avoid unsafe-inline:** Modify the CSP to exclude 'unsafe-inline' from script-src. Use safer alternatives like nonce or hash for inline scripts.

7. CSP: style-src unsafe-inline (Medium Risk, Medium Confidence)
https://a8462f3ftrial.launchpad.cfapps.us10.hana.ondemand.com

Description: The CSP includes 'unsafe-inline' for style-src, which may expose the application to style injection attacks.

Recommendation:

- **Avoid unsafe-inline:** Adjust the CSP to eliminate 'unsafe-inline' from style-src. Explore safer alternatives such as nonce or hash for inline styles.

These are the main vulnerabilities found about the URLs provided, more information can be found in the detailed report.

Conclusion for DAST Assessment

The Dynamic Application Security Testing (DAST) assessment has provided valuable insights into the runtime security of the application. Here are the key findings:

Cross-Domain Misconfiguration:

Identified a CORS misconfiguration on the web server, potentially exposing the application to cross-domain read requests. Immediate attention is needed to secure sensitive data access.

Cookie Security:

Uncovered a cookie with the SameSite attribute set to "none," posing a risk of being sent as a result of cross-site requests. Recommending an adjustment to enhance cookie security.

Third-Party Script Inclusion:

Noted the inclusion of third-party scripts, suggesting a potential avenue for security vulnerabilities. Advising a careful review of the necessity of these scripts and considering local hosting or trusted sources.

Content Security Policy (CSP):

CSP header not set, exposing the application to various attacks. Strongly recommending the implementation of CSP headers to mitigate risks, especially related to Cross-Site Scripting (XSS) and data injection.

CSP Directives:

Identified wildcard directives and 'unsafe-inline' usage in the CSP, increasing the attack surface. Recommending refinement of CSP directives for specific sources to minimize potential vulnerabilities.

Various Other Findings:

Uncovered miscellaneous issues such as Open Redirect, Missing Anti-clickjacking Header, Cookie without HttpOnly Flag, and Information Disclosure. These require tailored solutions based on each specific finding.

The DAST assessment has played a crucial role in evaluating the application's security posture during runtime. It has effectively identified vulnerabilities that could potentially be exploited by malicious actors. The process involved a comprehensive analysis of the application's behaviour in real-world scenarios, simulating various attack vectors.

Appendix - DAST

- ZAP Scanning Report – HTML document

SCA Assessment

Introduction to SCA Assessment

In the dynamic landscape of software development, managing external dependencies is critical to ensure the security and reliability of applications. Software Composition Analysis (SCA) emerges as a key practice, offering insights into the composition of software components and potential security risks associated with third-party libraries.

Semgrep, a leading security tool, has taken a significant step in enhancing dependency security. As of June 6, 2023, Semgrep Supply Chain is made freely accessible for all users, with a generous limit of up to 10 contributors. This tool efficiently scans dependencies, distinguishing false positives that are not reachable within the codebase, thereby saving valuable time.

Access and Environment Setup for SCA

Same as previous chapter SAST where we set up SEMGREP

SCA Methodology

https://semgrep.dev/blog/2022/a-deep-dive-into-semgrep-supply-chain/

Semgrep Supply Chain Workflow

1. Dependency Discovery

   Semgrep Supply Chain initiates the process by automatically identifying and compiling a comprehensive list of third-party dependencies used in an application.

2. Version Insight

   The tool provides insights into the versions of each dependency, allowing teams to monitor and manage version-specific vulnerabilities.

3. Vulnerability Detection

   Leveraging advanced reachability analysis, Semgrep Supply Chain identifies vulnerabilities in dependencies, focusing specifically on those that are reachable within the application's codebase.

4. Automated Rule Creation

   Semgrep's powerful polyglot engine is employed for the automated creation of high-confidence rules, ensuring swift adaptation to new vulnerabilities in publicly accessible databases.

5. Continuous Monitoring

   To stay proactive against emerging threats, Semgrep Supply Chain supports continuous scanning, allowing organizations to detect and address vulnerabilities promptly.

Techniques Employed by Semgrep Supply Chain

Reachability Analysis

Semgrep Supply Chain's unique approach to reachability analysis ensures that vulnerabilities detected are genuinely reachable within the application's codebase, minimizing false positives.

Automated Rule Generation

The tool leverages its powerful polyglot engine for the automated generation of rules based on emerging vulnerabilities, reducing the manual effort required for rule creation.

Context-Aware Rule Writing

Semgrep incorporates contextual clues like library imports and typed metavariables into its rules, enhancing accuracy and reducing false positives.

Taint Mode for Precision

Utilizing Semgrep's taint mode for rules ensures the direct usage of affected libraries and identifies impacted functionalities with precision.

External Data Integration

Semgrep Supply Chain seamlessly integrates with publicly accessible vulnerability databases, ensuring that organizations are well-informed about emerging threats.

Proof-of-Concept Extraction

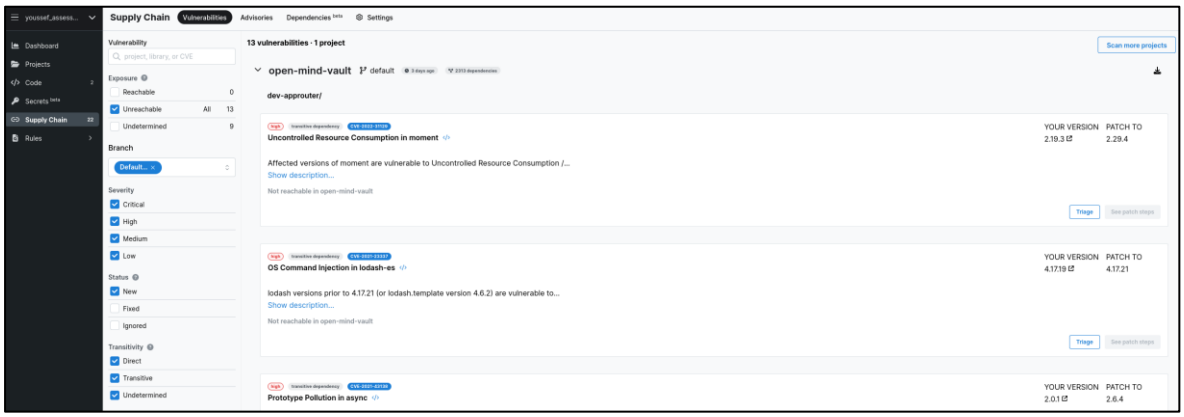
Heuristics within Semgrep are employed to extract proof-of-concept code from advisories and reference links, aiding in the validation of vulnerabilities.

Automated CVE Notifications

Semgrep automates the process of notifying users about new Common Vulnerabilities and Exposures (CVEs) from multiple sources across supported languages.

SCA Findings

After the SEMGREP scan I found following vulnerabilities of libraries used.



The scan has revealed there are zero reachable vulnerabilities, meaning it is not called by user code, which is verry good. Nine have been undetermined, where I would recommend updating the versions.

1. **CVE-2023-28155**
    - Medium severity
    - Transitive dependency: request
    - Vulnerability: Reliance on Uncontrolled Component in request leading to Server-Side Request Forgery (SSRF).
    - Your version: 2.88.2
    - Patch to: 3.0.0
    - Reachability analysis not available.
2. **CVE-2022-23540**
    - Medium severity
    - Transitive dependency: jsonwebtoken
    - Vulnerability: Reliance on Uncontrolled Component in jsonwebtoken leading to Improper Authentication.
    - Your version: 8.5.1
    - Patch to: 9.0.0
    - Reachability analysis not available.
3. **CVE-2023-26136**
    - Medium severity
    - Transitive dependency: tough-cookie
    - Vulnerability: Reliance on Uncontrolled Component in tough-cookie leading to Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution').
    - Your version: 2.5.0
    - Patch to: 4.1.3
    - Reachability analysis not available.

4. **CVE-2022-23541**
    - Medium severity
    - Transitive dependency: jsonwebtoken
    - Vulnerability: Reliance on Uncontrolled Component in jsonwebtoken leading to Improper Authentication.
    - Your version: 8.5.1
    - Patch to: 9.0.0
    - Reachability analysis not available.
5. **CVE-2022-23539**
    - Medium severity
    - Transitive dependency: jsonwebtoken
    - Vulnerability: Reliance on Uncontrolled Component in jsonwebtoken leading to Use of a Broken or Risky Cryptographic Algorithm.
    - Your version: 8.5.1
    - Patch to: 9.0.0
    - Reachability analysis not available.
6. **CVE-2022-25896**
    - Medium severity
    - Transitive dependency: passport
    - Vulnerability: Reliance on Uncontrolled Component in passport leading to Session Fixation.
    - Your version: 0.3.2
    - Patch to: 0.6.0
    - Reachability analysis not available.
7. **CVE-2023-26159**
    - Medium severity
    - Transitive dependency: follow-redirects
    - Vulnerability: Reliance on Uncontrolled Component in follow-redirects leading to Improper Input Validation.
    - Your version: 1.15.2
    - Patch to: 1.15.4
    - Reachability analysis not available.
8. **CVE-2023-45857**
    - Medium severity
    - Transitive dependency: axios
    - Vulnerability: Reliance on Uncontrolled Component in axios leading to Cross-Site Request Forgery (CSRF).
    - Your version: 0.26.1
    - Patch to: 1.6.0
    - Reachability analysis not available.

9. **CVE-2023-48631**
    - Medium severity
    - Transitive dependency: @adobe/css-tools
    - Vulnerability: Reliance on Uncontrolled Component in @adobe/css-tools leading to Inefficient Regular Expression Complexity.
    - Your version: 4.3.1
    - Patch to: 4.3.2
    - Reachability analysis not available.

To be completely secure the application should update the unreachable libraries as well.

1. **Dependency: moment**
    - **CVE-2022-31129**
    - **Version:** 2.19.3
    - **Patch to:** 2.29.4
    - *Description:* Uncontrolled Resource Consumption in moment. An attacker can provide inputs greater than 10k characters to the moment() constructor, resulting in noticeable slowdown. No reachability in open-mind-vault.
2. **Dependency: lodash-es**
    - **CVE-2021-23337**
    - **Version:** 4.17.19
    - **Patch to:** 4.17.21
    - *Description:* OS Command Injection in lodash-es. Versions prior to 4.17.21 are vulnerable to Command Injection via the template function. No reachability in open-mind-vault.
3. **Dependency: async**
    - **CVE-2021-43138**
    - **Version:** 2.0.1
    - **Patch to:** 2.6.4
    - *Description:* Prototype Pollution in async. Versions before 2.6.4 are vulnerable if user-controllable input is passed to async.mapValues(...). No reachability in open-mind-vault.
4. **Dependency: urijs**
    - **CVE-2021-3647**
    - **Version:** 1.16.1
    - **Patch to:** 1.19.7
    - *Description:* Open Redirect in urijs. Versions before 1.19.7 are vulnerable to URL Redirection To Untrusted Site ('Open Redirect'). No reachability in open-mind-vault.
5. **Dependency: qs**
    - **CVE-2022-24999**
    - **Version:** 6.5.2
    - **Patch to:** 6.5.3
    - *Description:* Prototype Pollution in qs. Affected versions are vulnerable to Improperly Controlled Modification Of Object Prototype Attributes ('Prototype Pollution'). No reachability in open-mind-vault.
6. **Dependency: moment**

- **CVE-2022-24785**
- **Version:** 2.19.3
- **Patch to:** 2.29.2
- *Description:* Path Traversal in moment. This vulnerability impacts npm (server) users of moment.js, especially if a user-provided locale string, eg fr, is directly used to switch moment locale. No reachability in open-mind-vault.

7. **Dependency: @sap/xssec**
    - **CVE-2023-49583**
    - **Version:** 3.0.9
    - **Patch to:** 3.6.0
    - *Description:* Improper Privilege Management in @sap/xssec. Affected versions are vulnerable to Improper Privilege Management / Authorization Bypass Through User-Controlled Key. No reachability in open-mind-vault.
8. **Dependency: json-schema**
    - **CVE-2021-3918**
    - **Version:** 0.2.3
    - **Patch to:** 0.4.0
    - *Description:* Improperly Controlled Modification of Dynamically-Determined Object Attributes in json-schema. json-schema 0.x before 0.4.0 is vulnerable to improperly controlled modification of dynamically-determined object attributes. No reachability in open-mind-vault.
9. **Dependency: debug**
    - **CVE-2017-16137**
    - **Version:** 4.1.1
    - **Patch to:** 4.3.1
    - *Description:* Uncontrolled Resource Consumption in debug. debug 3.x before 3.1.0 and debug 2.x before 2.6.9 are vulnerable to Uncontrolled Resource Consumption when untrusted user input is passed into the o formatter. No reachability in open-mind-vault.
10. **Dependency: @sap/xssec**
    - **CVE-2023-49583**
    - **Version:** 3.3.4
    - **Patch to:** 3.6.0
    - *Description:* Improper Privilege Management in @sap/xssec. Affected versions are vulnerable to Improper Privilege Management / Authorization Bypass Through User-Controlled Key. No reachability in open-mind-vault.
11. **Dependency: json5**
    - **CVE-2022-46175**
    - **Version:** 0.5.1
    - **Patch to:** 1.0.2
    - *Description:* Prototype Pollution in json5. The parse method of the JSON5 library before and including version 2.2.1 does not restrict parsing of keys named proto, allowing specially crafted strings to pollute the prototype of the resulting object. No reachability in open-mind-vault.

Conclusion for SCA Assessment:

The comprehensive SCA assessment using Semgrep Supply Chain has provided invaluable insights into the security landscape of external dependencies. Notably, the tool has successfully identified and analyzed third-party libraries, categorizing vulnerabilities into reachable and unreachable segments.

While the analysis revealed a commendable lack of reachable vulnerabilities, the importance of addressing the nine undetermined vulnerabilities is emphasized. It is recommended to promptly update the versions of the associated libraries to mitigate potential security risks and uphold a robust security posture.