OpenText™ Core Application Security Security Review

Tenant:	Nadro SAPI De CV
Application:	Mentorias-Front
Release:	1.0
Latest Analysis:	2025/10/28 02:41:57 AM
Latest Assessment Type:	Static+ Assessment

Executive Summary

Tenant:	Nadro SAPI De CV
Application:	Mentorias-Front
Release:	1.0
Business Criticality:	Low
SDLC Status:	QA/Test
Static Analysis Date:	2025/10/28
Dynamic Analysis Date:	---

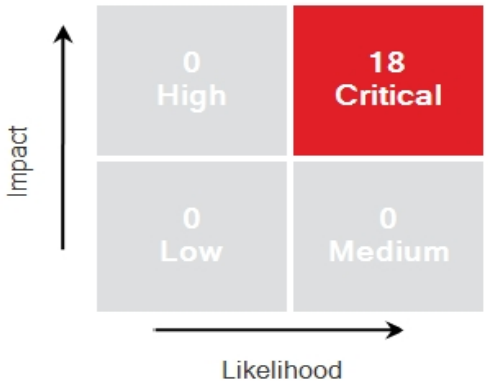
OpenText™ Core Application Security Security Rating

★☆☆☆☆

18 issues Status: Failed

Static: ✔	Dynamic: —
Open Source: —	Monitoring: —

Risk Totals by Severity

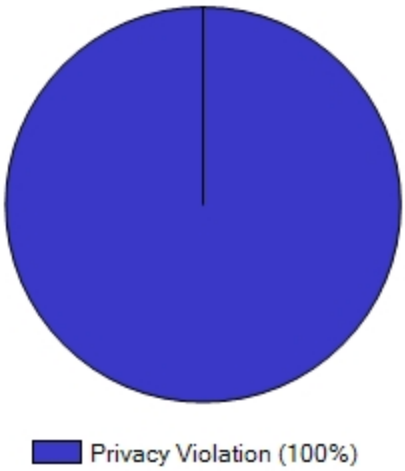


Issue Status

New	Existing	Reopened
0	18	0

Assignment Status

Most Prevalent Issues by Category



Developer Status

Auditor Status

Issue Breakdown

Issues are divided based on their impact (potential damage) and likelihood (probability of identification and exploit).

High impact / high likelihood issues represent the highest priority and present the greatest threat.

Low impact / low likelihood issues are the lowest priority and present the smallest threat.

See Appendix for more information.

Rating	Category	Test Type	
Critical	Privacy Violation	Static+ A...	18

Issue Breakdown by OWASP Top 10 2017
PCI Sections 6.3, 6.5 & 6.6

The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

The PCI compliance standards, particularly sections 6.3, 6.5, and 6.6, reference the OWASP Top Ten vulnerability categories as the core categories that must be tested for and remediated.

OWASP Category	Severity			
	Critical	High	Medium	Low
A1 - Injection				
A2 - Broken Authentication				
A3 - Sensitive Data Exposure	18			
A4 - XML External Entities (XXE)				
A5 - Broken Access Control				
A6 - Security Misconfiguration				
A7 - Cross-Site Scripting (XSS)				
A8 - Insecure Deserialization				
A9 - Using Components with Known …				
A10 - Insufficient Logging and Monito…				
Total	18			

Issue Breakdown by OWASP Top 10 2021
PCI Sections 6.3, 6.5 & 6.6

The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

The PCI compliance standards, particularly sections 6.3, 6.5, and 6.6, reference the OWASP Top Ten vulnerability categories as the core categories that must be tested for and remediated.

OWASP Category	Severity			
	Critical	High	Medium	Low
A01 - Broken Access Control				
A02 - Cryptographic Failures	18			
A03 - Injection				
A04 - Insecure Design				
A05 - Security Misconfiguration				
A06 - Vulnerable and Outdated Comp…				
A07 - Identification and Authenticatio…				
A08 - Software and Data Integrity Fail…				
A09 - Security Logging and Monitorin…				
A10 - Server-Side Request Forgery				
Total	18			

Issue Breakdown by Analysis Type

Issues are divided based on their impact (potential damage) and likelihood (probability of identification and exploit).

High impact / high likelihood issues represent the highest priority and present the greatest threat.

Low impact / low likelihood issues are the lowest priority and present the smallest threat.

See Appendix for more information.

Category	Static	Dynamic	Open S...	Monitor...
Privacy Violation	18	0	0	0
Total	18	0	0	0

Issue Detail

Below is an enumeration of all issues found in the project. The issues are organized by priority and category and then broken down by the package, namespace, or location in which they occur.

The priority of an issue can be Critical, High, Medium, or Low.

Issues from static analysis reported on at same line number with the same category originate from different taint sources.

7.1.1 Privacy Violation **Critical**

CWE-359
OWASP Top 10 2017: A3 - Sensitive Data Exposure
OWASP Top 10 2021: A02 - Cryptographic Failures
PCI 4.0.1: 3.3.1 Protect stored cardholder data, 3.3.2 Protect stored cardholder data, 3.3.3 Protect stored cardholder data, 3.5.1 Protect stored cardholder data, 4.2.2 Never send unprotected PANs by end-user messaging technologies, 8.3.1 Assign a unique ID to each person with computer access

Summary

The file **Login.jsx** mishandles confidential information on line **231**, which can compromise user privacy and is often illegal.Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.

Explanation

Privacy violations occur when:
1. Private user information enters the program.
In this case, the data comes from in **Login.jsx** on line **231**.

2. The data is written to an external location, such as the console, file system, or network.
In this case, the data is passed to in **Login.jsx** on line **231**.

Example 1: The following code stores user's plain text password to the local storage.

```
localStorage.setItem('password', password);
```

Although many developers treat the local storage as a safe location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information

- Accessed from a database or other data store by the application

- Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the

identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can create risk.

Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted. For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer email addresses to a spammer marketing an offshore gambling web site [1].

In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations:

- Safe Harbor Privacy Framework [3]

- Gramm-Leach Bliley Act (GLBA) [4]

- Health Insurance Portability and Accountability Act (HIPAA) [5]

- California SB-1386 [6]

Despite these regulations, privacy violations continue to occur with alarming frequency.

Execution

1. As part of any thorough audit for privacy violations, ensure that custom rules are written to identify all sources of private or otherwise sensitive information entering the program. Most sources of private data cannot be identified automatically. Without custom rules, your check for privacy violations is likely to be substantially incomplete.

Recommendation

When security and privacy demands clash, privacy should usually be given the higher priority. To accomplish this and still maintain required security information, cleanse any private information before it exits the program.
To enforce good privacy management, develop and strictly adhere to internal privacy guidelines. The guidelines should specifically describe how an application should handle private data. If your organization is regulated by federal or state law, ensure that your privacy guidelines are sufficiently strenuous to meet the legal requirements. Even if your organization is not regulated, you must protect private information or risk losing customer confidence.

The best policy with respect to private data is to minimize its exposure. Applications, processes, and employees should not be granted access to any private data unless the access is required for the tasks that they are to perform. Just as the principle of least privilege dictates that no operation should be performed with more than the necessary privileges, access to private data should be restricted to the smallest possible group.

References

1. AOL man pleads guilty to selling 92m email addies, J. Oates, https://www.theregister.co.uk/2005/02/07/aol_email_theft/

2. Privacy Initiatives, http://www.ftc.gov/privacy/
3. Safe Harbor Privacy Framework, https://2014-2017.commerce.gov/tags/safe-harbor-framework.html
4. Financial Privacy: The Gramm-Leach Bliley Act (GLBA), https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act
5. Health Insurance Portability and Accountability Act (HIPAA), http://www.hhs.gov/ocr/hipaa/
6. California SB-1386, https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=200120020SB1386
7. Writing Secure Code, Second Edition, M. Howard, D. LeBlanc
8. Fortify Taxonomy | Privacy Violation, https://vulncat.fortify.com/en/detail?category=Privacy+Violation

Instances

Privacy Violation — Critical

Package: N/A

Instance	Analysis Info	Analyzer
ID 289343775 - src/components/Auth/Login.jsx:231	**Sink:** Assignment to value in src/components/Auth/Login.jsx:231 **EnclosingMethod:** Login **Source:** Read confirmPassword from ..Login in src/components/Auth/Login.jsx:231	dataflow
ID 289343776 - src/components/Auth/Login.jsx:198	**Sink:** Assignment to value in src/components/Auth/Login.jsx:198 **EnclosingMethod:** Login **Source:** Read newPassword from ..Login in src/components/Auth/Login.jsx:198	dataflow
ID 289343777 - src/components/Auth/Login.jsx:338	**Sink:** Assignment to value in src/components/Auth/Login.jsx:338 **EnclosingMethod:** Login **Source:** Read formData.password from ..Login in src/components/Auth/Login.jsx:338	dataflow
ID 289343778 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read formData.password from ..Login in src/components/Auth/Login.jsx:338	dataflow
ID 289343779 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read errors.confirmPassword from ..Login in src/components/Auth/Login.jsx:252	dataflow
ID 289343780 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read completeNewPassword from . .completeNewPassword in src/services/api.js:71	dataflow
ID 289343781 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read newPassword from ..Login in src/components/Auth/Login.jsx:198	dataflow
ID 289343782 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read errors.newPassword from ..Login in src/components/Auth/Login.jsx:219	dataflow
ID 289343783 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read showConfirmPassword from ..onClick in src/components/Auth/Login.jsx:245	dataflow
ID 289343784 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read handleNewPasswordSubmit from ..Login in src/components/Auth/Login.jsx:177	dataflow
ID 289343785 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read errors.confirmPassword from ..Login in src/components/Auth/Login.jsx:251	dataflow
ID 289343786 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read errors.newPassword from ..Login in src/components/Auth/Login.jsx:218	dataflow
ID 289343787 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read showNewPassword from ..onClick in src/components/Auth/Login.jsx:212	dataflow
ID 289343788 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read errors.password from ..Login in src/components/Auth/Login.jsx:353	dataflow

Package: N/A		
Instance	Analysis Info	Analyzer
ID 289343789 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read errors.password from ..Login in src/components/Auth/Login.jsx:354	dataflow
ID 289343790 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read confirmPassword from ..Login in src/components/Auth/Login.jsx:231	dataflow
ID 289343792 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read showPassword from ..onClick in src/components/Auth/Login.jsx:347	dataflow
ID 289343793 - src/index.js:8	**Sink:** ~JS_Generic.render() in src/index.js:8 **EnclosingMethod:** ~file_function **Source:** Read completeNewPassword from . .handleNewPasswordSubmit in src/components/Auth/Login.jsx:133	dataflow

Static File Listing

The static file listing displays all files scanned by the SCA scanner.

Filename	Size (bytes)	Modified Date
/opt/Fortify/OpenText_SAST_Fortify_25.3.0/agentWork/w…	549	2025/10/28
package.json	991	2025/10/28
package-lock.json	693239	2025/10/28
postcss.config.js	83	2025/10/28
public/index.html	953	2025/10/28
src/App.jsx	3796	2025/10/28
src/components/Auth/Login.jsx	16666	2025/10/28
src/components/Dashboard/Dashboard.jsx	58114	2025/10/28
src/components/FormularioCapacitacion/FormularioCapacit…	34912	2025/10/28
src/components/FormularioConsulta/FormularioConsulta.jsx	36718	2025/10/28
src/data/areas.json	8339	2025/10/28
src/data/lugaresConsulta.json	163	2025/10/28
src/data/lugaresTrabajo.json	431	2025/10/28
src/data/motivosConsulta.json	762	2025/10/28
src/data/rangosEdad.json	45	2025/10/28
src/index.js	254	2025/10/28
src/services/api.js	5166	2025/10/28
src/services/cognitoAuth.js	7974	2025/10/28
src/utils/validation.js	800	2025/10/28
tailwind.config.js	1867	2025/10/28

Appendix - Descriptions of Key Terminology

Security Rating

The OpenText™ Core Application Security 5-star assessment rating provides information on the likelihood and impact of defects present within an application. A perfect rating within this system would be 5 complete stars indicating that no high impact vulnerabilities were uncovered.

Rating	
★☆☆☆☆	OpenText™ Core Application Security awards one star to applications that have undergone a security review that identifies critical (high likelihood and high impact) issues.
★★☆☆☆	OpenText™ Core Application Security awards two stars to applications that have undergone a security review that identifies no critical (high likelihood and high impact) issues. Vulnerabilities that are trivial to exploit and have a high business or technical impact should never exist in business-critical software.
★★★☆☆	OpenText™ Core Application Security awards three stars to applications that have undergone a security review that identifies no high (low likelihood and high impact) issues and meets the requirements needed to receive two stars. Vulnerabilities that have a high impact, even if they are non-trivial to exploit, should never exist in business critical software.
★★★★☆	OpenText™ Core Application Security awards four stars to applications that have undergone a security review that identifies no medium (high likelihood and low impact) issues and meets the requirements for three stars. Vulnerabilities that have a low impact, but are easy to exploit, should be considered carefully as they may pose a greater threat if an attacker exploits many of them as part of a concerted effort or leverages a low impact vulnerability as a stepping stone to mount a high-impact attack.
★★★★★	OpenText™ Core Application Security awards five stars, the highest rating, to applications that have undergone a security review that identifies no issues.

Likelihood and Impact

Likelihood

Likelihood is the probability that a vulnerability will be accurately identified and successfully exploited.

Impact

Impact is the potential damage an attacker could do to assets by successfully exploiting a vulnerability. This damage can be in the form of, but not limited to, financial loss, compliance violation, loss of brand reputation, and negative publicity.

OpenText™ Core Application Security Priority Order

Critical

Critical-priority issues have high impact and high likelihood. Critical-priority issues are easy to detect and exploit and result in large asset damage. These issues represent the highest security risk to the application. As such, they should be remediated immediately.

SQL Injection is an example of a critical issue.

High

High-priority issues have high impact and low likelihood. High-priority issues are often difficult to detect and exploit, but can result in large asset damage. These issues represent a high security risk to the application. High priority issues should be remediated in the next scheduled patch release.

Password Management: Hardcoded Password is an example of a high issue.

Medium

Medium-priority issues have low impact and high likelihood. Medium-priority issues are easy to detect and exploit, but typically result in small asset damage. These issues represent a moderate security risk to the application. Medium-priority issues should be remediated in the next scheduled product.

Path Manipulation is an example of a medium issue.

Low

Low-priority issues have low impact and low likelihood. Low-priority issues can be difficult to detect and exploit and typically result in small asset damage. These issues represent a minor security risk to the application. Low priority issues should be remediated as time allows.

Dead Code is an example of a low issue.

Issue Status

New

New issues are ones that have been identified for the first time in the most recent analysis of the application.

Existing

Existing issues are issues that have been found in a previous analysis of the application and are still present in the latest analysis.

Reopened

Reopened issues have been discovered in a previous analysis of the application but were not present in subsequent analyses. These issues are now present again in the most recent analysis of the application.