



Web Application Penetration Testing

October 20, 2026

This work was performed under contract to:

Northwind Logistics, Inc.

Chicago, IL, United States — illustrative sample report



Trusted by **Security Conscious** Companies

<https://iosentrix.com>

contact@iosentrix.com



Revision History

Version	Modification	Date	Author
1.0	Final report delivered	23/09/2026	ioSENTRIX

Document Confidentiality Statement

The information in this document is confidential to the person to whom it is addressed and should not be disclosed to any other person. It may not be reproduced in whole, or in part, nor may any of the information contained therein be disclosed without the prior consent of **ioSENTRIX LLC** ('the Company'). A recipient may not solicit, directly or indirectly (whether through an agent or otherwise) the participation of another institution or person without the prior approval of the directors of the Company.

Any form of reproduction, dissemination, copying, disclosure, modification, distribution and or publication of this material is strictly prohibited.

The information in this documentation is subject to change without notice and should not be construed as a commitment by the Company. The Company makes no representations or warranties, express or implied, with respect to the documentation and shall not be liable for any damages, including any indirect, incidental, consequential damages (such as loss of profit, loss of use of assets, loss of business opportunity, loss of data or claims for or on behalf of user's customers), that may be suffered by the use.

Table of Contents

	Revision History	2
	Document Confidentiality Statement	2
	Table of Contents.....	3
	Executive Summary	4
	Dashboard.....	5
1	Introduction	6
2	Methodology	6
2.1	Scope	6
2.1.1	Out of Scope	6
2.1.2	Severity rankings	6
3	Assessment Limitations	7
4	Technical Risks	9
4.1	Summary of findings	9
5	Findings.....	10
5.1	Critical Severity Risks.....	10
5.1.1	SQL Injection	10
5.1.2	Stored Cross-Site Scripting	12
5.2	High Severity Risks.....	14
5.2.1	Reflected Cross-Site Scripting (XSS)	14
5.2.2	Sensitive Data Disclosure	16
5.3	Medium Severity Risks	17
5.3.1	Sessions Do Not Expire on Password Change and Reset	17
5.3.2	Web Application Vulnerable to Session Fixation	19
5.3.3	HTML Injection	22
5.4	Low Severity Risks	24
5.4.1	Internal IP Disclosure through Canary Tokens	24
5.4.2	Groups Enumeration (IDOR).....	26
5.4.3	HTTP Strict Transport Security (HSTS) Not Implemented.....	29
5.5	Informational Severity Risks.....	30
5.5.1	Software Version Number Revealed	30
5.5.2	Information Leakage via JS Code	31
	About ioSENTRIX	32

Executive Summary

ioSENTRIX was engaged by Northwind Logistics, Inc. to assess the security of the Northwind Operations Portal, an internet-facing web application and backend API used to manage warehouse hubs, orders and customer deals. The objective was to identify vulnerabilities that an external attacker, or a low-privileged tenant user, could realistically exploit, and to rate the business risk of each.

Testing was performed from 8-19 September 2026 using a combination of manual, logic-aware testing and targeted automation, from the perspective of both an unauthenticated attacker and an authenticated low-privilege tenant user. The scope was limited to the application and API described in the Scope section.

ioSENTRIX identified twelve (12) vulnerabilities: two (2) Critical, two (2) High, three (3) Medium and three (3) Low, together with two (2) informational observations. The most serious issues allowed full compromise of the application database and takeover of other users' accounts, and are detailed in the sections that follow.

The Critical and High findings share a common root cause: user-supplied input is trusted and reflected into SQL queries, HTML responses and API payloads without consistent validation, output encoding or authorization checks.

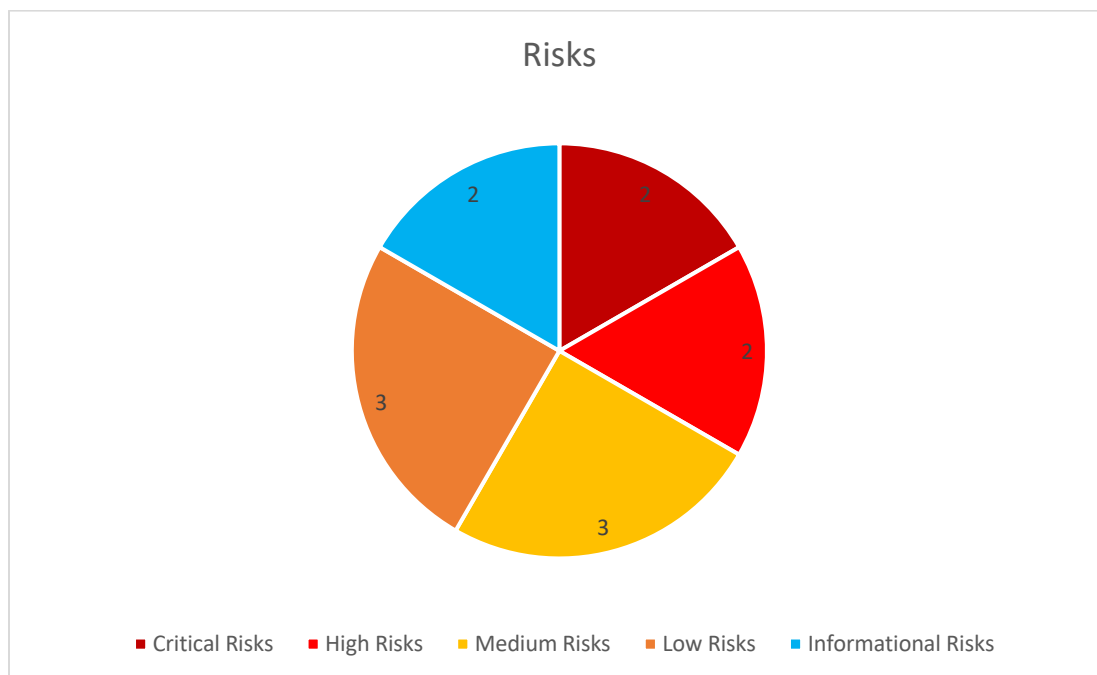
Our strategic recommendation is that Northwind integrate application security into the SDLC. Threat modeling, secure code review and a parameterized data-access layer would eliminate entire classes of the issues found here, rather than remediating them one at a time.

Dashboard

Name	Web Application Penetration Testing
Method	NIST Risk Severity Matrix
Dates	8-19 September 2026
Targets	Northwind Operations Portal (web application and API)

Findings Breakdown (Chart)

Critical Priority Risks	2
High Priority Risks	2
Medium Priority Risks	3
Low Priority Risks	3
Informational	2
Total Findings	12



1 Introduction

Northwind Logistics engaged ioSENTRIX to perform a security assessment of the Northwind Operations Portal and its backend APIs. Testing was conducted from 8-19 September 2026. The goal of this engagement was the discovery of security vulnerabilities, a qualitative assessment of the associated business risk, and actionable remediation guidance for each finding.

2 Methodology

2.1 Scope

The scope of the engagement was limited to the environment and the targets listed below:

Environment	Targets
Web application and API	https://ops-staging.northwind-demo.com (portal), https://api-staging.northwind-demo.com (API)

2.1.1 Out of Scope

The following components and tests were considered out of scope for this assessment:

- Any application or infrastructure outside the Northwind Operations Portal and its documented APIs, except where those systems were directly reachable from an in-scope component.
- Supporting policies, procedure and processes.
- Social engineering.
- Software Development life cycle
- Detailed hardware and software configurations

2.1.2 Severity rankings

ioSENTRIX ranks the severity of a risk using a method developed by the United States National Institute of Standards and Technology (NIST). Calculating risk severity as a derived value, based upon mitigation and impact assessments. The risk value represents the technical risk to the software and is calculated using the table below.

Impact						
Likelihood		Critical	High	Medium	Low	Informational
	Critical	Critical	High	Medium	Low	Informational
	High	Critical	High	Medium	Low	Informational
	Medium	High	Medium	Medium	Low	Informational
	Low	Medium	Low	Low	Low	Informational
	Informational	Low	Low	Informational	Informational	Informational

Table 1: NIST Risk Severity Matrix

In the above table, the Likelihood is defined as the probability of a given threat exploiting the vulnerability. That incorporates ease of exploitation and determination and capability of the given threat agent. The impact is defined as a possible impact of occurrence of a risk.

The levels of risk severity have the following meanings:

- **Critical:** A critical risk is characterized by a very high likelihood and a high or very high impact. Remediation must be prioritized immediately and scheduled as the highest priority. Critical issues are considered blockers for new releases and must be resolved before deployment.
- **High:** High-severity issues present a substantial risk, though they may not block new releases. Remediation should be scheduled promptly and treated as a high priority.
- **Medium:** Medium severity issues require remediation. The organization should create and implement a remediation plan to address these risks effectively.
- **Low:** For low-severity issues, the organization may choose to either mitigate the risk or formally accept it, depending on its risk tolerance and strategic priorities.
- **Informational:** Informational findings have minimal or no direct impact on the target scope. However, they are reported as they may contribute to risk under specific circumstances, such as when combined with other vulnerabilities or technologies. Remediation is optional.

3 Assessment Limitations

The ever-changing technology landscape and the increasing sophistication of attacks against networked systems are reasons for which no entity can truthfully claim to identify all the security issues, nor guarantee the lifetime-security of an organization's network and applications. Note that this point-in-time assessment was based on the best-effort and was performed only on the environment provided by Client. Thus, changes to the environment may impact the applicability of the results provided herein.

ioSENTRIX cannot guarantee 100% coverage for any security assessment.

4 Technical Risks

4.1 Summary of findings

Below is the summary of the vulnerabilities found during the assessment. Complete details are available in the later section of this report.

No	Security Risk	Severity	Status
5.1.1	SQL Injection	Critical	Open
5.1.2	Stored Cross-Site Scripting	Critical	Open
5.2.1	Reflected Cross-Site Scripting (XSS)	High	Open
5.2.2	Sensitive Data Disclosure	High	Open
5.3.1	Sessions Do Not Expire on Password Change and Reset	Medium	Open
5.3.2	Web Application Vulnerable to Session Fixation	Medium	Open
5.3.3	HTML Injection	Medium	Open
5.4.1	Internal IP Disclosure through Canary Tokens	Low	Open
5.4.2	Groups Enumeration (IDOR)	Low	Open
5.4.3	HTTP Strict Transport Security (HSTS) Not Implemented	Low	Open
5.5.1	Software Version Number Revealed	Informational	Open
5.5.2	Information Leakage via JS Code	Informational	Open

5 Findings

5.1 Critical Severity Risks

5.1.1 SQL Injection

Risk Rating	Impact	Likelihood
Critical	Critical	High

Description:

The hub-management API builds SQL statements by concatenating values taken directly from the request body into the query string. Because the database cannot distinguish the developer's intended query structure from attacker-supplied text, any SQL syntax placed in those values is parsed and executed as part of the query.

The portal lets operations staff create and update warehouse hubs. We found that the country field sent to the hub-update endpoint is concatenated into an UPDATE statement without parameterization. By injecting a sub-select we were able to read arbitrary database content - including the live database version - and from there any data reachable by the application's database account. This represents a full read/write compromise of the application database and is rated Critical.

Instances:

1. **PATCH** <https://api-staging.northwind-demo.com/api/v3/hubs/{hubId}>

Steps to Reproduce:

1. Send a PATCH request to the hub-update endpoint and replace the value of the country field with a SQL sub-select that returns the database version.

```
PATCH /api/v3/hubs/NW-HUB-014 country = (select version())
```
2. The database version string is returned in the stored hub record, confirming that the injected expression was executed by the database.

Evidence:

Figure 1: Database version returned in the injected parameter

Remediation:

Rewrite all SQL queries constructed through dynamic concatenation to use an injection-safe query mechanism such as prepared statements with parameterized queries. Most modern programming languages provide a feature called “parameterized queries” that allow user-supplied data to be inserted safely as values in dynamic SQL queries. Rather than construct the dynamic SQL query by concatenating user-supplied data to static SQL query string fragments, data values are identified in the query by parameter markers or variables. Dynamic data is then passed through a mechanism provided by SQL that prevents the supplied data from changing the meaning of the query.

Note: the exact syntax and use of prepared statements with parameterized queries varies from language to language. The following links provide general guidance for secure SQL query construction in .NET, Java, PHP:

- <https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.htm>
- <http://php.net/manual/en/mysql.prepare.php>
- [https://msdn.microsoft.com/en-us/library/bb738521\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/bb738521(v=vs.100).aspx)

5.1.2 Stored Cross-Site Scripting

Risk Rating	Impact	Likelihood
Critical	Critical	High

Description:

The Operations Portal lets a user record Order Details that are later displayed to any other user who opens the order. We found that the content of the Order Details field is stored and then rendered back to other users without output encoding.

A stored cross-site scripting (XSS) flaw occurs when an application saves attacker-supplied input and later returns it to other users' browsers in a context where it is executed as script rather than displayed as text. Because the payload is stored server-side, every user who opens the affected order is attacked automatically - no phishing or user interaction is required.

This gives the attacker control of the HTML and JavaScript running in the victim's authenticated session: they can read or alter what the victim sees, perform actions as the victim, or steal the victim's session and act as that user for its lifetime. Because operations and admin staff routinely open orders raised by others, a single poisoned order can escalate to an administrator.

Instances:

1. <https://ops-staging.northwind-demo.com/orders>

Steps to Reproduce:

1. Open the site and navigate to Order Details
2. In the Order Details field, submit a short script-element proof-of-concept payload and save the order.
3. Observe that pop up will appear which shows the result of above script
4. Note down the Order id and Login as a different user
5. As a different user, open the same order by its Order ID.
6. Observe that the JavaScript popup will be displayed

Evidence:

Figure 2: Stored payload executing in another user's session

Remediation:

Stored Cross-Site Scripting (XSS) is prevented by encoding data before inserting it into the generated web page. Each character of the data is encoded and the result string is then inserted onto the generated web page. This technique of encoding values before inserting them on the web page is called "Output Encoding". Output Encoding libraries exist for most popular programming languages and frameworks.

A web page has seven different output contexts and each output context requires a different encoding scheme. Data must be encoded using the proper scheme. The seven different encoding schemes are:

- HTML Text Element
- HTML Attribute
- URL Parameter
- JavaScript Literal
- HTML Comment
- HTTP Header
- CSS Property

For example, the characters: <, >, ", ' are encoded as <, >, ", ' for when those characters are inserted into an HTML Text Element. When those characters are inserted as a URL Parameter, the same characters are encoded as %3C, %3E, %22, %27.

Libraries for implementing the encoding schemes exist for most popular programming languages.

- OWASP Java Encoder: Java only
- Microsoft Web Protection Library: .NET languages
- Ruby - escapeHTML() - only supports HTML Text Encoding
- Jgencoder in JQuery: for preventing DOM-based XSS

5.2 High Severity Risks

5.2.1 Reflected Cross-Site Scripting (XSS)

Risk Rating	Impact	Likelihood
High	Critical	Medium

Description:

The document-preview feature reflects a user-supplied value - the document name - into the HTML response without encoding it. An attacker who persuades a victim to open a crafted link, or to preview a document whose name contains markup, has script of their choosing executed in the victim's session.

Unlike the stored flaw above, this payload travels in the request, so exploitation typically relies on the victim opening an attacker-supplied link; the impact within the victim's session is the same.

Instances:

1. <https://app-staging.northwind-demo.com/documents/view>
2. <https://app-staging.northwind-demo.com/files/preview>

Steps to Reproduce:

1. Log into your account.
2. Craft a document whose file name contains a short HTML script element.
3. Click on Upload File button.
4. Select the malicious PDF file.
5. Upload the File.
6. Preview the uploaded document.
7. You will see an XSS popup.

Evidence:

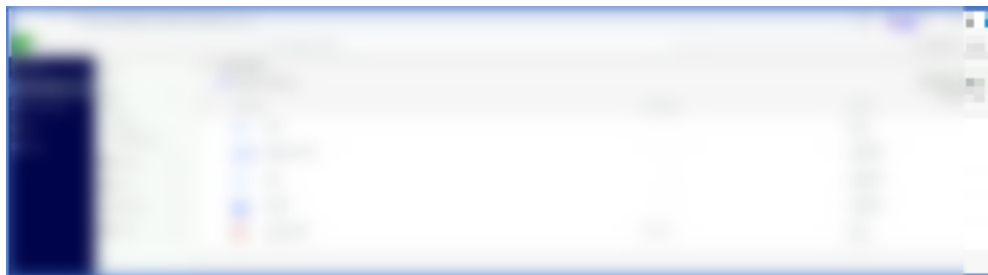


Figure 3: File-upload feature used to deliver the reflected payload

Remediation:

Reflected cross-site scripting is prevented by contextually encoding every piece of user-supplied data before it is written into the response. Each HTML context - element text, attribute, URL parameter, JavaScript literal and CSS - requires its own encoding scheme.

The technique of encoding values before inserting them on the web page is called "Output Encoding". Output Encoding libraries exist for most popular programming languages and frameworks.

A web page has seven different output contexts, and each output context requires a different encoding scheme. Data must be encoded using the proper scheme. The seven different encoding schemes are:

- HTML Text Element
- HTML Attribute
- URL Parameter
- JavaScript Literal
- HTML Comment
- HTTP Header
- CSS Property

Libraries for implementing the encoding schemes exist for most popular programming languages.

- OWASP Java Encoder: Java only
- Microsoft Web Protection Library: .NET languages
- Ruby - `escapeHTML()` - only supports HTML Text Encoding
- Jgencoder in JQuery: for preventing DOM-based XSS
- Google Capabilities based JavaScript CAJA
- OWASP JXT- automatically encodes string data with the proper encoding

For this application specifically, serve previewed documents with a Content-Type of `application/octet-stream` and a Content-Disposition of `"attachment"` so the browser downloads rather than renders them.

5.2.2 Sensitive Data Disclosure

Risk Rating	Impact	Likelihood
High	High	High

Description:

A low-privileged user can retrieve a full directory of every user on the platform - across all tenants - by calling the users API directly. The response includes names, email addresses, internal user identifiers and the deals each user is associated with. This breaks tenant isolation and exposes customer and employee data that an ordinary user should never see.

Because the endpoint returns the complete list in a single call and applies no tenant-level authorization check, an attacker needs only one valid account to harvest the entire user base - useful for targeted phishing, credential stuffing and competitive intelligence.

Instances:

1. <https://api-staging.northwind-demo.com/api/v3/users>

Steps to Reproduce:

1. Authenticate to the application.
2. Visit the instance mentioned above.
3. Parse the returned JSON in order to compile a list of users.

Evidence:

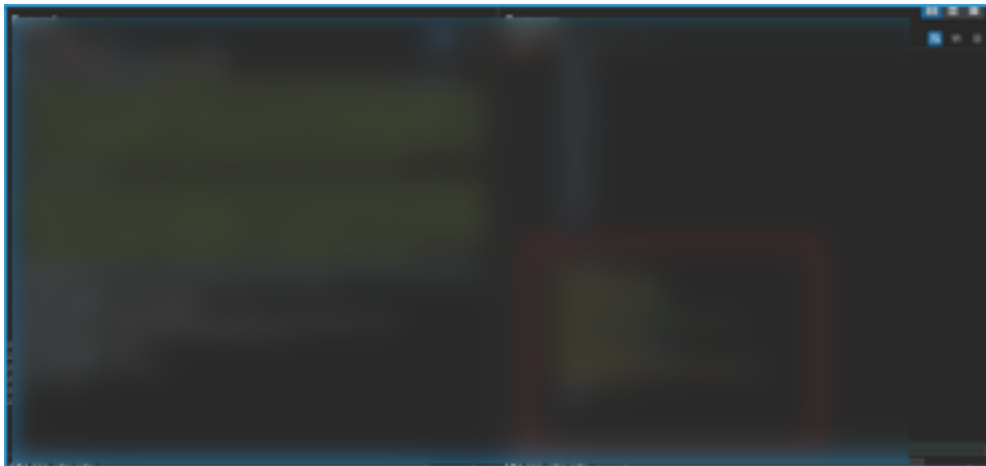


Figure 4: API response disclosing other tenants' user records

Remediation:

The application should limit the information returned in the response or limit this API endpoint for admin access only.

5.3 Medium Severity Risks

5.3.1 Sessions Do Not Expire on Password Change and Reset

Risk Rating	Impact	Likelihood
Medium	High	Medium

Description:

The portal lets users change or reset their password, but existing sessions remain valid afterwards. If an attacker has captured a user's session - or the user changed their password precisely because they suspected compromise - the old session continues to work, defeating the most common containment step available to the user.

An attacker who has obtained a session can retain access indefinitely simply by keeping the session active, even after the legitimate user changes their credentials.

Instances:

1. <https://ops-staging.northwind-demo.com/account/password>

Steps to Reproduce:

1. Initiate authentication workflow in two different browser sessions.
2. Change the password in browser number 1.
3. Refresh browser number 2 to assure it has not logged out.

Evidence:

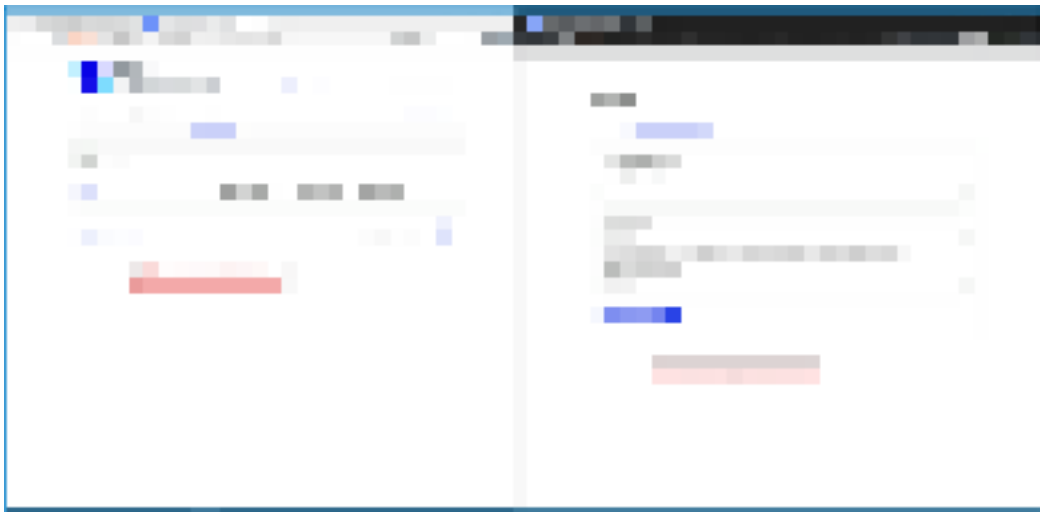


Figure 5: Session still valid after a password reset

Remediation:

Upon a password reset or update, the web application must terminate all the active sessions associated with the user's account. Additionally, an absolute session timeout (such as 24 hours) should be coupled with inactivity timeout.

5.3.2 Web Application Vulnerable to Session Fixation

Risk Rating	Impact	Likelihood
Medium	Medium	Medium

Description:

The application does not issue a new session identifier when a user authenticates: the identifier present before login is still valid afterwards. An attacker who plants a known session identifier in a victim's browser and then waits for the victim to log in inherits the victim's authenticated session.

Session Fixation is a type of session-hijacking vulnerability that happens when an application does not update the session identifier when a user authenticates or has a change of permission. An attacker persuades a valid user to authenticate with a session identifier provided or already compromised by the attacker. Upon successful authentication, the server assigns the new level of permission to the attacker-controlled session identifier, allowing the attacker full access to the victim's account.

For a Session Fixation vulnerability to exist, the following sequence of events needs to occur:

1. A victim has their session identifier set (or fixed) by an attacker by either of the following ways depending on the server behavior:
 - a. The server fails to issue a new session token upon the user's permission level changes (such as when a user is authenticated to the application): A previously generated by the server session identifier can be used by an attacker to hijack the session if the attacker obtains that session identifier. OR
 - b. When the server accepts any random identifier for the session: Attackers can persuade victims to use a session identifier value of the attacker's choice due to missing server-side validation.
2. The victim's privilege level is changed with the attacker's session identifier (for example, the victim authenticates to the application).
3. The session identifier is unchanged after changes in privilege level, hence establishing a valid user session with the attacker's session identifier. For example, same session identifier is used before and after the user's authentication.

Instances:

1. <https://ops-staging.northwind-demo.com/login>

Steps to Reproduce:

1. Configure a browser to use an intercepting proxy, then open <https://ops-staging.northwind-demo.com/login>.
2. Login by entering user credentials and navigate to Home Page
3. Intercept the request and response with Burp Suite and locate the ASP.NET_SessionId in Proxy Http History tab

4. Logout from the web application and login again with same credentials
5. Intercept request and response with Burp Suite and locate the ASP.NET_SessionId in Proxy Http History tab. ASP.NET_SessionId value from last session is sent again with login request and the server validates the same old ASP.NET_SessionId with successful response

Evidence:

Figure 6: Pre-authentication session identifier accepted after login

Remediation:

When authenticating a user, the application must invalidate the existing session identifier, and create a new session identifier with the updated session data. Session fixation remediation is a two-steps process:

- Session identifiers must be regenerated and re-issued after any changes to the privilege level of a user. For example, when a user authenticates with the application successfully, the privilege level changes from unauthenticated to authenticated, and hence the application must invalidate any existing session identifiers and regenerate a new session identifier. Other cases when such a regeneration may be necessary include (but not limited to) changes in user's password, authorization level, or permissions
- Session identifier in every incoming request must be validated by the server using a lookup to ensure the client-submitted value matches the value generated by the server during user's authentication, and that the value is in line with the user's current level of authorization

5.3.3 HTML Injection

Risk Rating	Impact	Likelihood
Medium	Medium	High

Description:

The portal renders attacker-supplied input as HTML in two places - the message attached to an upload request, and uploaded file names surfaced in notification emails. In both cases the markup is rendered by the recipient's client rather than escaped.

An attacker can inject arbitrary HTML - for example a link that looks like the legitimate "Upload" button but points to a site they control - into content that other users and administrators are expected to trust.

A user can also perform HTML Injection by changing a file name to HTML injection payload and share the document with the admin.

Instances:

1. <https://ops-staging.northwind-demo.com/uploads/request>

Instance 1:**Steps to Reproduce:**

1. Log into your account.
2. Create a new folder.
3. Open the folder and click on Send Upload Request button.
4. In the Message box, enter HTML markup - for example a heading followed by a link element pointing to an attacker-controlled site - and submit the form.
5. Send the form.

Instance 2:**Steps to Reproduce:**

1. Start an intercepting proxy and log in to <https://ops-staging.northwind-demo.com>.
2. Enable intercepting and upload a file.
3. Change the filename to xss payload for an example : `<img src=x>`
4. The portal emails other users a notification that includes the attacker-controlled file name.
5. Payload will be triggered in the mail.

Evidence:

Figure 7: Message field rendering attacker-supplied HTML

Remediation:

Encode all user-supplied data for the HTML context before rendering it in pages or emails, and apply a restrictive Content-Security-Policy. Treat uploaded file names as untrusted input and escape them wherever they are displayed.

5.4 Low Severity Risks

5.4.1 Internal IP Disclosure through Canary Tokens

Risk Rating	Impact	Likelihood
Low	Low	Medium

Description:

The portal processes uploaded files server-side. We uploaded a benign tracking ("canary") document and, when the application opened it, the resulting callback revealed an internal IP address from Northwind's infrastructure. This leaks details of the internal network to anyone able to upload a file.

The issue occurs when canary tokens are triggered by an internal user or process. Instead of only alerting administrators discreetly, the token also reveals the internal IP address from where it was accessed. This information could be inadvertently included in the alert or log files that are generated when the canary token is activated.

Instances:

1. <https://ops-staging.northwind-demo.com/deals/{dealId}/files>

Steps to Reproduce:

1. Generate a "docx" canary file.
2. Create a deal through the application.
3. Access the deal's "Files" tab.
4. Upload the canary file.
5. The file will be triggered and the IP will be sent in the notification.

Evidence:

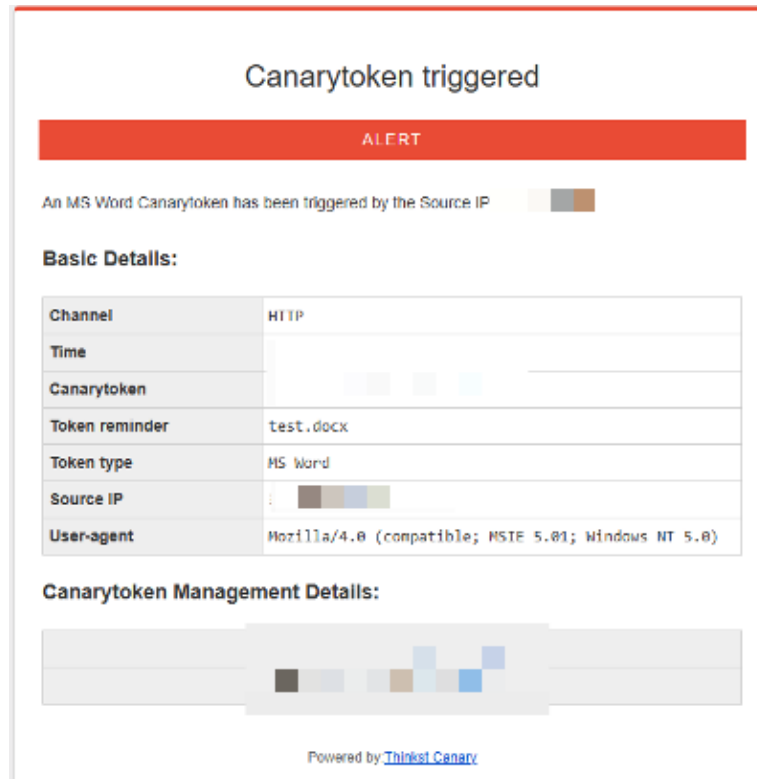


Figure 8: Internal IP address disclosed via an uploaded canary file

Remediation:

- Update the alerting system to filter out any internal IP addresses from being displayed or logged.
- Implement a logging policy that excludes sensitive information, such as internal IP addresses, from being recorded.
- Also running the uploaded files in a sandboxed environment that prevents initiating call back connections will help address this issue.

5.4.2 Groups Enumeration (IDOR)

Risk Rating	Impact	Likelihood
Low	Low	Medium

Description:

The group-details API accepts a group identifier and returns that group's data with no check that the caller's tenant owns the group. A regular user in one tenant can read groups belonging to any other tenant simply by supplying its identifier, and there is no rate limiting to slow enumeration.

The UUID endpoint, designed to provide information about a specific group, lacks proper authorization checks. An attacker with a valid UUID can exploit this vulnerability to access details such as accountId, library, isEnabled, name, modifiedWhen, groupNumber, createdWhen and status of any group within the application. Furthermore, the absence of rate-limiting protection facilitates attackers in conducting brute-force attacks to enumerate different UUIDs, potentially leading to unauthorized access to a large volume of group information.

Instances:

1. <https://api-staging.northwind-demo.com/api/web/v3/groups/{groupId}>

Steps to Reproduce:

1. Log into an Admin account create a new group.
2. Click on the newly created group and copy group-id from URL.
3. Log into a regular account of a different tenant/company.
4. Add the cookies of regular user in the below HTTP Request.

```
GET /api/web/v3/groups/{Group-ID} HTTP/1.1
Host: api-staging.northwind-demo.com
Cookie: YOUR_COOKIES_HERE
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.3
```

5. Replace {Group-ID} with the Admin Group ID.
6. Send the Request through Burp Suite.
7. Review the Group Information in the Response.

Evidence:

Figure 9: Creating a group and copying its identifier in the Admin tenant

Figure 10: A different tenant's user retrieving the group via IDOR

Remediation:

To mitigate IDOR (Insecure Direct Object Reference) vulnerabilities, you can implement the following remediation measures:

1. **Implement Access Controls:** Apply proper access controls to restrict users from accessing resources they are not authorized to view or modify. Validate user permissions and authorization levels before granting access to sensitive information or performing actions.
2. **Use Indirect Object References:** Avoid directly exposing internal identifiers or sensitive data in URLs or client-side references. Instead, use indirect references or tokens that are mapped to the actual resources on the server side. This way, even if an attacker modifies the parameter or URL, they won't be able to access unauthorized resources.
3. **Perform Server-Side Authorization and Validation:** Always validate and authorize user requests on the server side. Never rely solely on client-side controls, as they can be bypassed or modified. Validate user input, parameters, and access rights to ensure that users are authorized to perform requested actions and access specific resources.
4. **Implement Role-Based Access Control (RBAC):** Utilize RBAC mechanisms to define and enforce granular access control policies based on user roles, permissions, and responsibilities. Assign appropriate access levels to users and ensure that access controls are enforced consistently across the application.

5.4.3 HTTP Strict Transport Security (HSTS) Not Implemented

Risk Rating	Impact	Likelihood
Low	Medium	Low

Description:

The server does not implement the "HTTP Strict-Transport Security" (HSTS) web security policy mechanism. When HSTS is enabled, the web application sends a special response header, "Strict-Transport-Security" to the client with a duration of time specified. Once a supported browser receives this header, that browser will only make requests to the application over HTTPS for the duration of time specified in the header. Any links to resources over HTTP will be rewritten to HTTPS before the request is made.

Applications that do not utilize the "HTTP Strict-Transport Security" policy are more susceptible to man-in-the-middle attacks via SSL stripping, which occurs when an attacker transparently downgrades a victim's communication with the server from HTTPS to HTTP. Once this is accomplished, the attacker will gain the ability to view and potentially modify the victim's traffic, exposing sensitive information and gaining access to unauthorized functionality.

Instances:

1. <https://ops-staging.northwind-demo.com/>

Steps to Reproduce:

1. Configure system to use burp as an intercepting proxy.
2. Enable interception in burp proxy → intercept tab.
3. Enter web address in browser and login with test credentials.
4. Go to Home tab and intercept the request

Evidence:



Figure 11: Response missing the Strict-Transport-Security header

Remediation:

The application server should send the "Strict-Transport-Security" HTTP header in each response indicating that future requests to the domain use only HTTPS. The following is a basic example of the HSTS HTTP header, setting a max-age of one year: `Strict-Transport-Security: max-`

age=31536000 Subdomains should also be configured in this manner, by including the "includeSubDomains" flag:

```
<pre>Strict-Transport-Security: max-age=31536000; includeSubDomains; </pre>
```

5.5 Informational Severity Risks

5.5.1 Software Version Number Revealed

Risk Rating	Impact	Likelihood
Informational	Informational	Informational

Description:

The server software version used by the application is revealed by the web server. Displaying version information of software information could allow an attacker to determine which vulnerabilities are present in the software, particularly if an outdated software version is in use with published vulnerabilities.

Instances:

1. <https://ops-staging.northwind-demo.com/>

Steps to Reproduce:

1. Configure browser to use Burp Suite as a proxy. Open the URL from instances section and log into the application.
2. Intercept the request and view the response to verify disclosure of software and its version

Evidence:

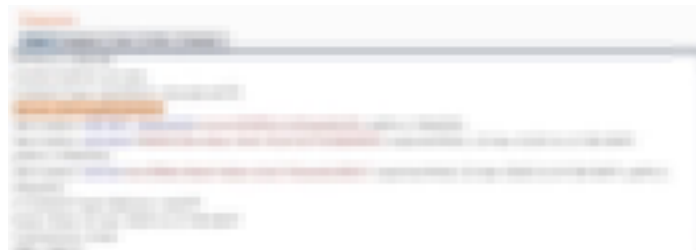


Figure 12: Response header disclosing the server software and version

Remediation:

Verbose server information should be removed from all HTTP responses. This can be performed by modifying the server's configuration files or through the use and configuration of a web application firewall.

5.5.2 Information Leakage via JS Code

Risk Rating	Impact	Likelihood
Informational	Informational	Informational

Description:

A client-side JavaScript bundle served by the application contains sensitive values - test-account credentials and default SSO PINs - in clear text. Anyone who loads the application can read these directly from the browser, with no authentication required.

Instances:

1. <https://app-staging.northwind-demo.com/static/js/app.bundle.js>

Steps to Reproduce:

1. Visit the affected endpoint.
2. Search the file for strings such as "password" or "@northwind-demo.com".
3. Multiple test accounts and default SSO pins will be revealed.

Evidence:



```

    }, 500);
    this.post(
      const data = JSON.parse(request.requestBody);
      if (data.password === "12345" (
        return (
          token: _ssoHelper.default.generateToken(),
          session_key: "12345",
          tfa_status: "disabled"
        );
      ) else {
        return new _response.default(400, {}, {
          non_field_errors: ["Bad credentials"]
        });
      }
    );
    this.post("
    return (0, _authTestHelpers.getMockDatabaseToken) ();
  );

```

Figure 13: Credentials hard-coded in a client-side JavaScript file

Remediation:

JavaScript files must be stripped from any kind of sensitive information so that they are useful for application code, but do not expose any sensitive user or application information. Credentials, IP addresses, PII, internal email addresses, identifiers and other unnecessary information must never be included in code.

About ioSENTRIX

ioSENTRIX LLC is a Security Consulting firm now proudly accredited by CREST, an internationally recognized hallmark for professional cybersecurity services. As a CREST accredited pentesting company, we adhere to the highest standards of service, ensuring that our clients receive the most comprehensive and reliable security testing available.

We provide a wide range of security consulting services to clients worldwide. Our clientele includes Fortune 500 companies, large enterprises, small start-ups, financial institutions, and several high-tech companies. At ioSENTRIX, we are committed to delivering innovative cybersecurity solutions.

As an innovative consulting company, we offer a full range of cyber security services tailored to businesses of all sizes and budget requirements. Our focus is on helping clients identify, mitigate, and prevent vulnerabilities in their software, infrastructure, and cloud environments.

Our comprehensive vulnerability assessment includes design-review, threat modeling, penetration testing, code review, and open source software security. We are equipped with the necessary tools and expertise to secure your business, allowing you to focus on growth and innovation.

Learn more about how our CREST accreditation and our services can benefit your business at <https://www.iosentrix.com>.

ioSENTRIX LLC.

13800 Coppermine Rd, Suite 190
Herndon Virginia 20171 (USA)

Sales: 1 (888) 958-0554
Email: sales@iosentrix.com