Web Application Penetration Testing

December 16, 2024

This work was performed under contract to:

Sample Report.
New York, United States

ioSENTRIX

Trusted by Security Conscious Companies

https://iosentrix.com
contact@iosentrix.com

Revision History

Version	Modification	Date	Author
0.1	Acme Inc – Pentest	12/06/2024	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

Executive Summary

ioSENTRIX was engaged by client to perform the penetration testing of their Web application. The goal of this engagement was the discovery of security vulnerabilities and a qualitative assessment of the associated risks levels.

ioSENTRIX conducted manual and automated testing against the Web application to discover the vulnerabilities. The scope of this assessment was limited to the Web application only. The exact scope is listed in the Scope section of this report.

Overall, ioSENTRIX identified twelve (12) security vulnerabilities. Of those, two (02) were ranked as Critical severity, two (02) were ranked as High severity, three (03) were ranked as Medium severity and three (03) were ranked as Low severity. Additionally, two (2) informational findings were identified that highlight potential gaps in the adherence to the best policy practices for web applications and backend APIs. These risks and corresponding recommendations are detailed in Sections below.

The Critical and High severity vulnerabilities mainly involved the lack of parameterized queries and unencrypted communication.

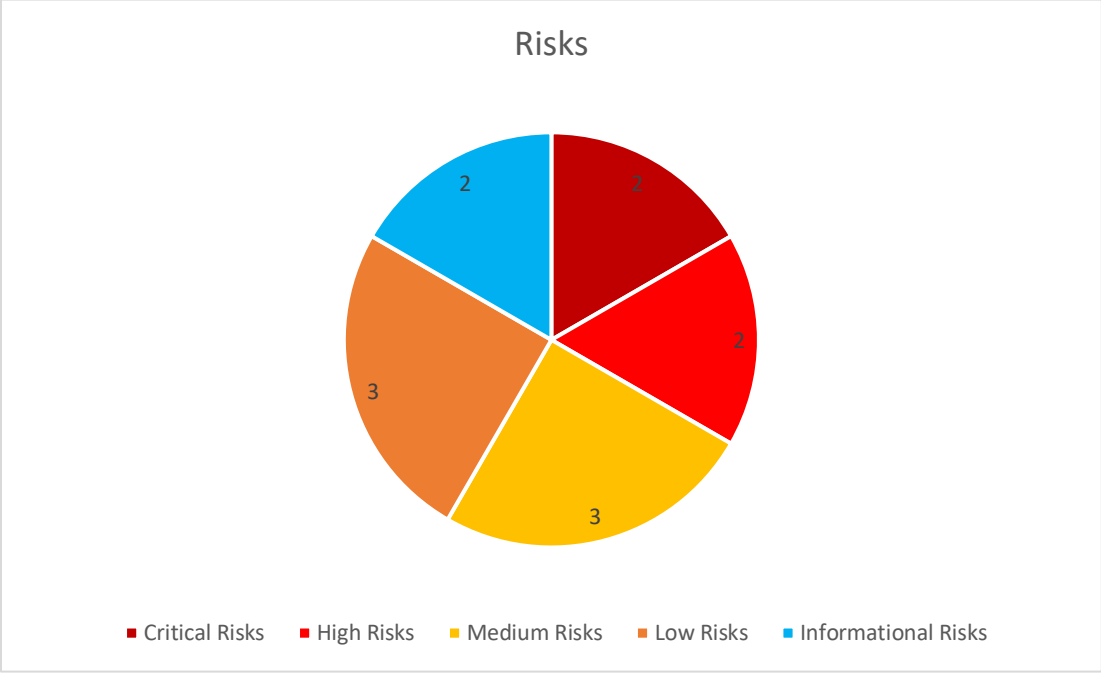
Our Strategic recommendation is that Client should integrate application/software security into the SDLC process. Architecture Review, Threat Modeling, and Secure Code Review can further reduce the attack surface significantly and help in keeping customer's data secure.

Dashboard

Name	**Web Application Penetration Testing**
Method	NIST Risk Severity Matrix
Dates	July 12th, 2018 to July 26th, 2018
Targets	Web Application Pentest

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

Client engaged ioSENTRIX to perform the security assessment of their Operations portal and backend APIs. Testing was conducted from July 12th, 2018 to the July 26th, 2018. The goal of this engagement was the discovery of security vulnerabilities and a qualitative assessment of the associated risks levels. Remediation strategies are provided for each vulnerability to help client mitigate or reduce the identified risks.

2 Methodology

2.1 Scope

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

Environment	Targets
Web Application	https://abcd.com

2.1.1 Out of Scope

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

- Any application and infrastructure external to Operations Portal. In cases where the Operations Portal had inbound and/or outbound interfaces with another application, the interfaces and communications were considered in scope. All other external elements were excluded.
- 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.

Likelihood \ Impact	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:** The likelihood is very high and the impact is either high or very high, therefore the mitigation should be scheduled as soon as possible with a highest priority. Additionally, Critical issues must be considered as blocker for the new release
- **High:** Although High severity issues are not blockers, they may pose significant risk. Mitigation should be scheduled as soon as possible with a higher priority.
- **Medium:** Mitigation is necessary. The organization should develop a remediation plan.
- **Low:** The organization must decide whether to correct the problem or accept this risk.
- **Informational:** Informational vulnerabilities have little-or-no impact to the target scope by themselves. They are included however, as they may be a risk when combined with other circumstances or technologies not currently in place. Remediation is not necessary.

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
4.1.1	SQL Injection	Critical	Open
4.1.2	Stored Cross-Site Scripting	Critical	Open
4.2.1	Reflected Cross-Site Scripting (XSS)	High	Open
4.2.2	Sensitive Data Disclosure	High	Open
4.3.1	Sessions Do Not Expire on Password Change and Reset	Medium	Open
4.3.2	Web Application Vulnerable to Session Fixation	Medium	Open
4.3.3	HTML Injection	Medium	Open
4.4.1	Internal IP Disclosure through Canary Tokens	Low	Open
4.4.2	Groups Enumeration (IDOR)	Low	Open
4.4.3	HTTP Strict Transport Security (HSTS) Not Implemented	Low	Open
4.5.1	Software Version Number Revealed	Informational	Open
4.5.2	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 application executes SQL queries generated by dynamically concatenating user supplied data to static SQL query strings. In this implementation, the database has no way of distinguishing between the developer's intended SQL syntax included from user-supplied input. As a result, any user-supplied data containing SQL syntax inserted into the static query will be interpreted and executed. This leaves the application vulnerable to SQL injection since an attacker can inject SQL query syntax that intentionally alters the target query's structure.

The web application allows a user to pick, receive, shelve and transit the orders. We discovered that parameter "xyz" in hubs is vulnerable to SQL Injection attack and could be used to execute arbitrary SQL statements on the vulnerable system.

Instances:

1. http://abc.stag.ABCD/api/pthubone

Steps to Reproduce:

1. Run the curl command to verify the SQL Injection vulnerability in parameter

```
curl -i -s -k  -X $'PATCH' \
    -H  $'Host:  hubs.stag.ABCD'  -H  $'Content-Length:  272'  -H  $'Accept:
application/json,  text/plain,  */*'  -H  $'Origin:  http://operations.stag.ABCD'  -H
$'Authorization: Bearer Sample804a5958-1233-4c1b-9595-aaaa5e5c9a26' -H $'User-Agent:
Mozilla/5.0  (X11;  Linux  x86_64)  AppleWebKit/537.36  (KHTML,  like  Gecko)
Chrome/62.0.3202.89 Safari/537.36' -H $'Content-Type: application/json' -H $'Referer:
http://operations.stag.ABCD/hubs/pthu Sample bone/edit' -H $'Accept-Encoding: gzip,
deflate' -H $'Accept-Language: en-US,en;q=0.9' -H Sample $'Connection: close' \
    --data-binary  $'{\"hubId\"  Sample:\"pthubone\",\"name\":\"  Sample  abcd  PT  Hub
one\",\"country\":\"usa\",\"city\":\"abc\",\"addressLine\":\"asdf\',hub_info.name  =
\'abcd  PT  Hub  one\',  hub_info.country  =  (select  version())  where
hub_info.hub_id=\'pthubone\'
##\",\"latitude\":24.7136,\"longitude\":46.6753,\"active\":true}' \
    $'http://hubs.stag.ABCD/api/pthubone'
```

2. You will get the following response with the database version used at the backend

Evidence:

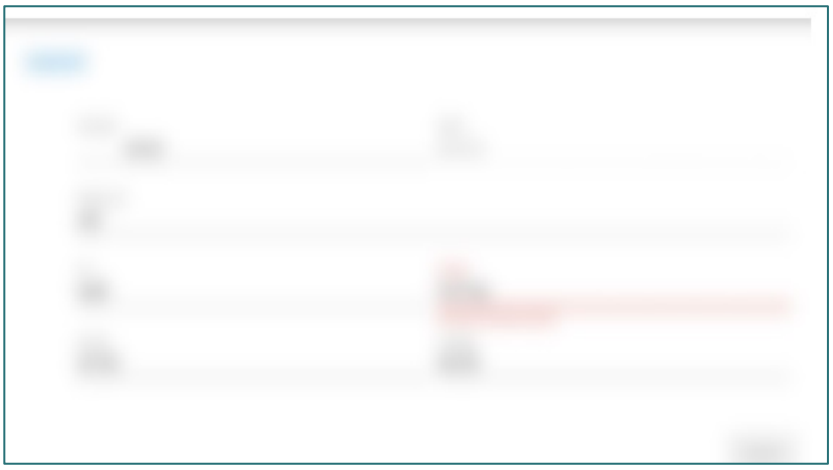


Figure 1: Reflected database version in parameter

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/mysqli.prepare.php
- 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 portal allows a user to enter the Order Details that can be viewed by the other users. We found that the parameter "xyz" in the request is vulnerable to Stored Cross-Site Scripting attack.

A Stored Cross-Site Scripting (XSS) vulnerability occurs when a web application sends stored strings that were provided by an attacker to a victim's browser in such a way that the browser executes part of the string as code. The string contains malicious data and is initially stored server-side, often in the application's database. The application later retrieves the malicious data and inserts it into a web page. This results in the victim's browser executing the attacker's code within a legitimate user's session.

Stored Cross-Site Scripting vulnerabilities give the attacker control of HTML and JavaScript running the user's browser. The attack can alter page content with malicious HTML or JavaScript code. The attacker can arbitrarily alter page content displayed to the victim and can execute application functions using the victim's application identity if the victim is authenticated to the application. An often-cited example use of a Stored Cross-Site is where the attacker sends himself/herself the victim's session identifier. With this session identifier, the attacker can then perform application functions using that user's identity for the duration of that session.

Instances:

1. http://test.abc.com

Steps to Reproduce:

1. Open the site and navigate to Order Details
2. In the OrderDetail field, write script "><script>alert(1)</script> and press the update button
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. Browse the order by using Order id and look type in the noted Order id
6. Observe that the JavaScript popup will be displayed

Evidence:

Figure 2: Stored XSS script executed successfully

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:

A Reflected Cross-Site Scripting (XSS) vulnerability occurs when a web application sends strings that were provided by an attacker to a victim's browser in such a way that the browser executes part of the string as code. The string contains malicious data and is passed as to the application through a parameter that an attacker can control (e.g. a URL parameter or an HTML form field). The application immediately inserts it into its response. This results in the victim's browser executing the attacker's code within a legitimate user's session. Attackers typically exploit reflected XSS vulnerabilities by sending users malicious links containing JavaScript code (e.g. via e-mail) or by posting malicious code to other sites that the vulnerable application's users may visit.

The application reflects the User Input in the URL path into web page without any encoding which allows the creation of HTML tags.

Instances:

1. http://xyz.stag.abcd.com
2. http://xyz.stag.abcd.com

Steps to Reproduce:

1. Log into your account.
2. Download the malicious PDF file: https://www.acmeinc.com/file/
3. Click on Upload File button.
4. Select the malicious PDF file.
5. Upload the File.
6. Open the uploaded pdf file.
7. You will see an XSS popup.

Evidence:

Figure 3: Upload File button.

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.

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

In Context of this Application Set PDF Content-type to application/octet-stream and Set Content-Disposition Header to “attachment” when previewing PDF files.

5.2.2 Sensitive Data Disclosure

Risk Rating	Impact	Likelihood
High	High	High

Description:

A High security flaw has been identified in a web application. This vulnerability exposes sensitive data, including a complete list of organization users, their involvement in various deals, and other confidential information like the user id. Such exposure poses significant risks to privacy, operational security, and the organization's integrity, potentially leading to legal consequences.

An attacker can abuse this design to compile a list of valid users through automated brute force guessing attempts. Simple scripts can be found or written that automatically guess usernames by submitting them to the login function with a dummy password and observing the server's responses.

Instances:

1. http://xyz.stag.abcd.com

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 2: Listing other user's data

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 web application allows the users to change or reset their account password. We discovered that application fails to expire or invalidate previously logged in sessions once the user updates the account password. If the application does not have session expiration on password change or reset, then the compromised credential can be used indefinitely by just ensuring the session doesn't timeout from inactivity.

An attacker with compromised credentials can continuously access the victim's account by remaining active on the web application. The process of bypassing inactivity timeout can be automated through browser plugins or via Selenium scripts.

Instances:

1. http://xyz.stag.abcd.com

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:

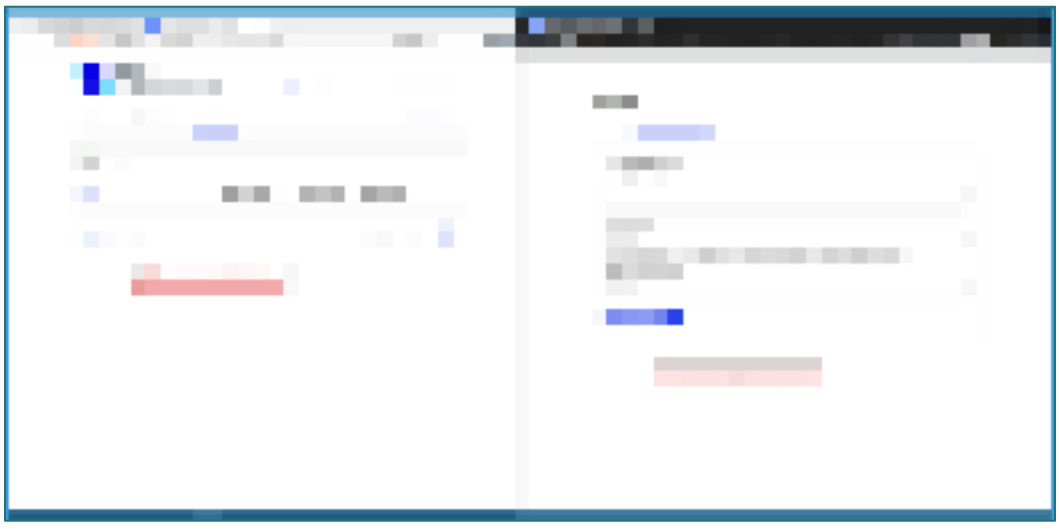


Figure 3: Non expired session after password reset.

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:

We observed that web application suffers from session fixation vulnerability.

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. http://xyz.stag.abcd.com

Steps to Reproduce:

1. Configure a browser to use Burp Suite as a proxy. Then open the URL https://xyz.stag.abcd.com
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 4: Successful response with old session

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:

An HTML injection vulnerability occurs when a web application sends strings that were provided by an attacker to a victim's browser in such a way that the browser renders part of the string as HTML markup. The string contains malicious data and is passed as-is to the application through a parameter that an attacker can control (e.g. a URL parameter or an HTML form field). The application immediately inserts it into its response. This results in the victim's browser rendering the attacker-controlled HTML markup.

The application allows users to send an Upload Request along with a Message, and it has been discovered that the Message content is susceptible to HTML injection. This vulnerability exposes users to potential attacks and could lead to various security risks such as sending malicious links to users instead of legit Upload form link.

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://op.stag.abc

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. Enter email in the form and in the Message Box enter HTML tags. <h1>Click on the Upload Link below</h1>
Upload Link

5. Send the form.

Instance 2:
Steps to Reproduce:

1. Start burpsuite and Login to https://xyz.com
2. Enable intercepting and upload a file.
3. Change the filename to xss payload for an example : <img src=x>
4. Acme send a mail for new documents update.
5. Payload will be triggered in the mail.

Evidence:

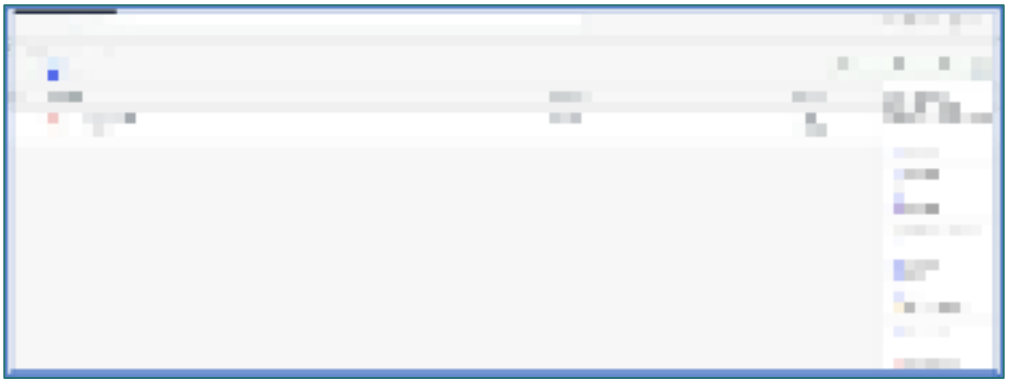


Figure 5: Click on Send Upload Request (Instance 1).

Remediation:

HTML injection 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 into the generated web page.

5.4 Low Severity Risks

5.4.1 Internal IP Disclosure through Canary Tokens

Risk Rating	Impact	Likelihood
Low	Low	Medium

Description:

A security vulnerability involving the unintentional disclosure of internal IP addresses through the use of canary tokens. Canary tokens are a security tool used to track unauthorized access or breaches by embedding a unique code in various locations within a system. When accessed or triggered, these tokens alert administrators to potentially malicious activity. However, in this specific vulnerability, the implementation of canary tokens has led to the inadvertent exposure of internal IP addresses.

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://op.stag.abc

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:

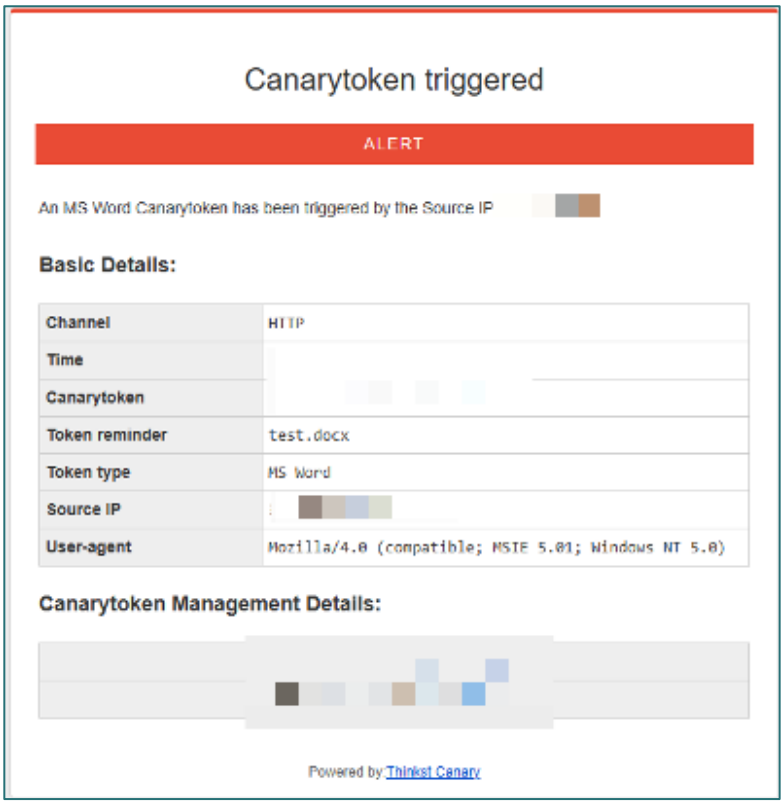


Figure 6: Internal IP Disclosed via Canary Tokens

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 application is susceptible to Insecure Direct Object Reference (IDOR) attack, allowing an attacker with a valid UUID to retrieve information about any group. Additionally, there is a lack of rate-limiting protection, enabling attackers to enumerate different UUIDs without restriction.

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://op.stag.abc

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: abc.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:

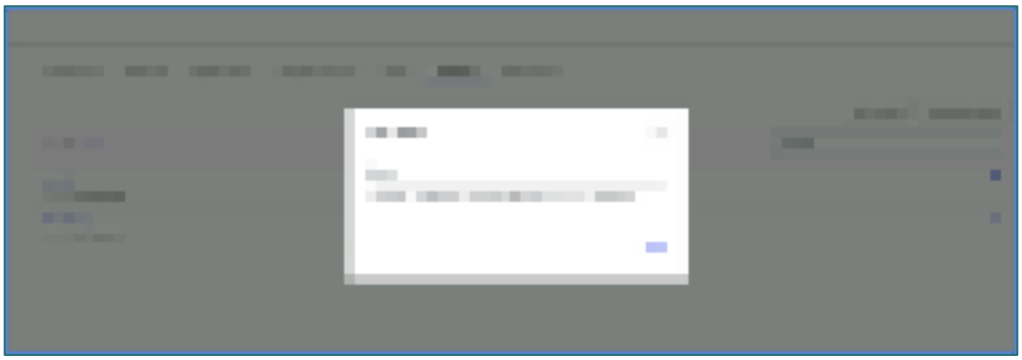


Figure 7: Creating group in Admin account

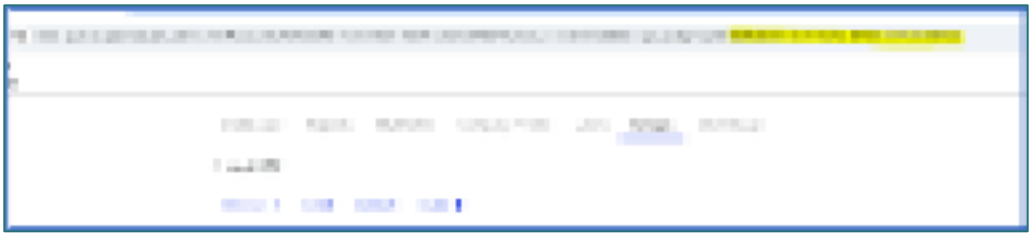


Figure 8: Copy the Group ID

Figure 9: Copy the regular user Cookies of a different tenant

Figure 10: Unauthorized Group Access

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	High	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://op.stag.abc

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
5.

Evidence:

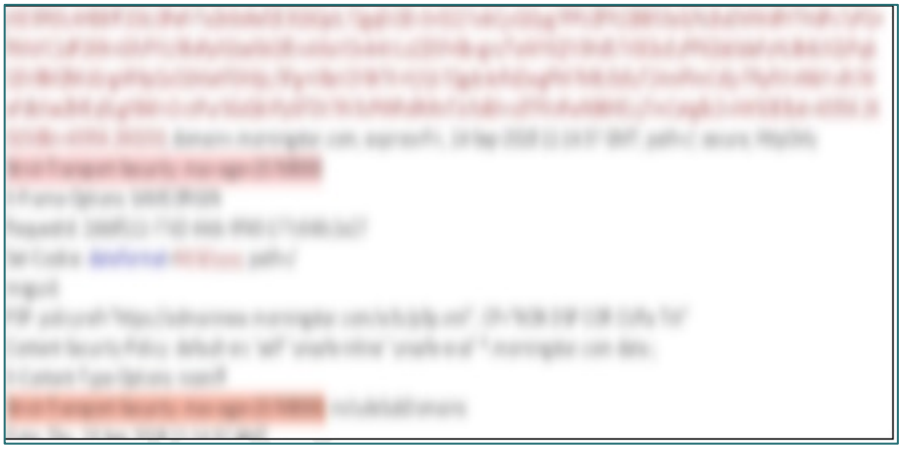


Figure 11: HTTP Strict Transport Security (HSTS) Not Implemented

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://op.stag.ABC

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 revealing software 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:

JavaScript files in the application source code contain sensitive information. While JS files are useful for front end tasks, sensitive information may be exploited by users authorized to access the code, or leaked to unauthorized individuals who gain access to the code base. Examples of sensitive information commonly found in code comments include (but are not limited to) test credentials, internal email addresses, sample data that illustrates data formats, and account numbers, IP addresses for internal subsystems, etc. JS files may also give away information about the application's business logic and control flow that would otherwise not be obvious to someone simply reading the code. For example, a JS file may point out a bug that is not fixed, or provide information about application logic that could not be discerned by reading the HTML code alone.

Instances:

1. https://op.stag.ABC

Steps to Reproduce:

1. Visit the affected endpoint.
2. Search for the word “Password” or “@domain.com”.
3. Multiple test accounts and default SSO pins will be revealed.

Evidence:

Figure 13: Hardcoded Passwords in JavaScript

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