Security Assessment Report

[REDACTED]

Prepared For

[REDACTED]

[DATE REDACTED]

1. Executive Summary

Between [REDACTED] and [REDACTED], OSec, LTD conducted a security assessment of [REDACTED] web and mobile applications. The purpose of this engagement was to identify potential security weaknesses that could affect the confidentiality, integrity, or availability of the application and its users. The testing uncovered several areas for improvement related to access control, session handling, and client-side protections.

One of the key findings involved access control issues, where OSec was able to access other users' account data, such as names and balances, by manipulating identifiers in API requests. In addition, the Multi-Factor Authentication (MFA) flow allowed for full API access prior to completing the second authentication step, which reduces the effectiveness of the control in preventing unauthorised access. These issues highlight opportunities to strengthen user data protection and session enforcement.

The assessment also noted the absence of rate limiting on functions such as profile updates and message sending, which could allow authenticated users to perform these actions in rapid succession without restriction. In another area, full bank account numbers and phone numbers were returned in API responses, even though they were masked in the user interface. This creates a scenario where sensitive information may be more accessible than intended if a session is compromised.

Additional observations included the ability to determine whether specific user accounts exist by observing differences in API error messages, as well as a file upload feature that accepted PDFs with embedded JavaScript which executed upon opening. The application's Content Security Policy (CSP) also included relaxed script directives that may allow broader script execution than necessary.

Based on the results of this assessment, OSec recommends that [REDACTED] review access control enforcement across all APIs, ensure that MFA is required before granting full session access, and implement rate limiting to discourage automated use. Sensitive data returned in API responses should align with what is shown in the user interface, and responses should be standardised to prevent account enumeration. Enhancing the CSP policy and applying stricter controls around file uploads will also help strengthen protection against client-side threats. Making these updates will support a more resilient security posture for the platform and its users.

2. Scope

The assessment scope consisted of the following component(s):

- Authenticated Web Application Assessment
- Android and iOS Mobile Application Assessment

The following hosts and locations were included as part of the scope:

Targets / Hosts
[REDACTED]

3. Rules of Engagement

The OSec team adhered to the following rules of engagement:

1. No brute force attacks were undertaken. Given the capabilities of most modern operating systems to implement an account lockout feature, no attempt was made to brute force any passwords.
2. No Denial-of-Service (DoS) attacks were launched.

4. Findings Summary

The vulnerabilities identified during this assessment have been categorised by their estimated remediation urgency into the following categories: Now or Later. Table 1 below defines these categories.

Remediation Urgency	Definition	Impact
Now	Issue must be remediated immediately.	Exploitation and/or compromise of data in the short term is likely and imminent.
Later	Issue should be remediated at some point.	Exploitation and/or compromise of data in the medium to long term is possible.

Table 1 - Vulnerability Ratings

Based on the findings during the assessment, nine (9) vulnerabilities were identified. Tables 2 through 5 below summarise the results of testing and vulnerabilities according to type, remediation urgency, and count.

Testing Area	Vulnerabilities Present	Critical Data Accessed
Web Application	Yes	No
Android Application	Yes	No
IOS Application	Yes	No

Table 2 - Breach Summary

#	Vulnerability Name/Description	Remediation Urgency	Risk Level
1	MFA Bypass	Now	High
2	Insecure Direct Object Reference (IDOR)	Now	Medium
3	Bypass of Session Security Controls	Now	Medium
4	Sensitive Information Disclosure	Later	Medium
5	Account Enumeration Using Application Error Messages	Later	Medium
6	Unsanitised File Upload	Now	Medium
7	No Rate Limit Implemented	Later	Informational
8	Unauthenticated Access To Authenticated Functions	Later	Informational
9	Insecure Content Security Policy Header	Later	Informational

Table 3 - List of Issues

Remediation Urgency	Total Number of Vulnerabilities
Now	4
Later	5
Total	**9**

Table 4 - Number of Vulnerabilities by Remediation Urgency

Risk Level	Total Number of Vulnerabilities
Critical	0
High	1
Medium	5
Low	0
Informational	3
Total	**9**

Table 5 - Number of Vulnerabilities by Risk Level

The tables below summarise the vulnerabilities discovered and exploited.

1 - MFA Bypass	
Location/s	[REDACTED]
Remediation Urgency	Now
Risk Level	High
CVSS 4.0 Score & Vector	7.6 (CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N)
CVE/CWE	CWE-287: Improper Authentication

Issue Description

The application allows authenticated users to bypass the MFA process by reusing the issued authorisation token before completing MFA verification. During testing, after submitting valid credentials and reaching the MFA challenge screen, the application returns an authentication token that could be used to interact with authenticated API endpoints without completing the second authentication step. This indicates that the backend fails to properly enforce MFA requirements before granting access to protected resources, significantly undermining the intended security of the MFA flow.

Steps to Reproduce

1. Attempt to login to the application while proxying HTTP requests using a tool such as Burp Suite.
2. Once at the MFA screen, check the proxied traffic and observe the application is sending valid API requests with a JWT token.
3. Using the JWT token, attempt to send other API requests, observe that they

succeed.

Impact
An attacker who compromises a user's credentials could gain full access to the user's account and sensitive data without needing to complete MFA, defeating one of the primary defences against unauthorised access. This exposes the application to a range of attacks including data theft, account takeover, and privilege misuse. The issue significantly reduces the effectiveness of MFA and places all accounts at elevated risk if credentials are ever leaked or phished.
Remediation
Ensure that access to all authenticated resources is blocked until MFA has been fully completed. Authorisation tokens issued prior to MFA completion should either be scoped for limited use (e.g., only MFA-related endpoints) or invalidated if used outside the MFA flow. Introduce strict server-side checks to validate the MFA status of a session before allowing access to any authenticated functionality.
Associated / Relevant Evidence
[REDACTED] Figure 1 - Presented with the MFA Screen After Login [REDACTED] Figure 2 - Background Request Made when Visiting the MFA Screen Showing Working JWT [REDACTED] Figure 3 - Using the JWT Token on a Separate API Endpoint, Without Restriction
References
https://cwe.mitre.org/data/definitions/287.html

2 - Insecure Direct Object Reference (IDOR)	
Location/s	[REDACTED]
Remediation Urgency	Now
Risk Level	Medium
CVSS 4.0 Score & Vector	5.3 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N)
CVE/CWE	CWE-639: Authorization Bypass Through User-Controlled Key

Issue Description

The application suffers from widespread instances of Insecure Direct Object References (IDOR) across both the mobile and web platforms. By modifying object identifiers such as account numbers or user IDs in API requests, it is possible to access sensitive data belonging to other users. This included viewing account balances, full names, and other personal information without proper authorisation checks. These vulnerabilities indicate a lack of server-side enforcement of object-level access control, allowing users to access resources they do not own simply by guessing or manipulating identifiers.

Steps to Reproduce

Web

1. Login to the application authenticated while using a HTTP proxy such as Burp Suite to capture background requests.
2. Navigate through the application to hit various endpoints mentioned above and in the Appendix.
3. Modify the ID identified within the URL to something outside of the account's normal control, this will usually be either a [REDACTED], [REDACTED]or a

[REDACTED].

4. In the identified cases, if a correct ID of another user's account, order, or document is entered, that information should then be correctly returned in the response. An account's total value is returned (Figure 4), and a document is returned as a base 64 string that can be decoded to get the original file contents (Figure 5). Also, orders can be cancelled simply by enumerating a guessable ID (Figure 7-9).

Mobile

1. Login to the mobile application while using a HTTP proxy such as Burp Suite to capture background requests.
2. Use one of the identified vulnerable endpoints to modify the ID value to target a different user's account.
3. Observe that the application returns information associated with the target account such as the user's [REDACTED], [REDACTED], and [REDACTED] performance (Figures 10-11).

Impact

IDOR vulnerabilities can lead to significant exposure of sensitive user data, including personal details. In this case, unauthorised access to other users' account balances and names poses a serious privacy risk and may lead to account profiling, fraud, or regulatory violations. The presence of multiple IDORs across different parts of the application increases the risk of large-scale data exposure and undermines trust in the platform's data security controls.

Remediation

Implement strict server-side access control checks to verify ownership or authorisation before returning any resource or object. Never rely solely on client-side logic or obscured identifiers to control access. Perform a comprehensive audit of all endpoints that accept user-controllable identifiers and ensure that proper authorisation checks are in place. Incorporate automated testing to detect IDOR patterns during development and regression testing cycles.

Associated / Relevant Evidence
[REDACTED] Figure 4 - (Web) Retrieving [REDACTED] [REDACTED] Figure 5 - (Web) Retrieving the base64 Encoded String of Another User's [REDACTED] [REDACTED] Figure 6 - (Web) Decoding the base64 Encoded File to See its Contents [REDACTED] Figure 7 - (Web) [REDACTED] [REDACTED] Figure 8 - (Web) Deleting [REDACTED] [REDACTED] Figure 9 - (Web) [REDACTED] by a Different User Account [REDACTED] Figure 10 - (Mobile) Viewing Another User's Account Information After Changing ID [REDACTED] Figure 11 - (Mobile) Viewing Another User's [REDACTED] After Changing ID
References
https://cwe.mitre.org/data/definitions/639.html

3 - Bypass of Session Security Controls	
Location/s	[REDACTED]
Remediation Urgency	Now
Risk Level	Medium
CVSS 4.0 Score & Vector	5.3 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N)
CVE/CWE	CWE-693: Protection Mechanism Failure

Issue Description

The application implements a session security mechanism designed to protect users by terminating their session if they attempt to access disallowed or potentially malicious functionality. This mechanism is triggered by redirecting the user to a logout endpoint, effectively forcing re-authentication. However, during testing, it was found that this protection can be bypassed by programmatically ignoring or preventing the redirect. By sending requests directly to sensitive endpoints and avoiding the redirect flow, the session remains active and no logout occurs. This undermines the intended control and allows potentially unsafe actions to continue under an authenticated context.

Steps to Reproduce

1. Visit the application in an authenticated session.
2. Use a proxy such as Burp Suite to capture background HTTP requests and prevent the redirect.
3. Attempt to access a potential malicious path that will trigger the defence such as /shell.exe.
4. Using Burp Suite, capture the application redirect to [REDACTED] and drop the

request.

5. Revisit the site and observe the user is authenticated and the session wasn't revoked.

Impact

This behaviour allows an attacker who knows how the mechanism operates to intentionally bypass security-triggered logout events and continue using the session without interruption. If the control was designed to protect against exposure to specific high-risk actions or known abuse vectors, bypassing it may enable scenarios such as continued access to deprecated or restricted functionality, malicious file delivery, or replaying restricted flows. The control becomes ineffective if it depends solely on a frontend-triggered redirect rather than server-enforced session invalidation.

Remediation

Enforce session termination directly on the server-side when a violation or policy condition is met, rather than relying on client-side redirect behaviour to initiate logout. Any action deemed risky or outside of accepted behaviour should trigger backend-driven session invalidation that cannot be bypassed by omitting client behaviour. Additionally, monitor and log such access attempts for review and response.

Associated / Relevant Evidence

[REDACTED]

Figure 12 - Visiting a Targeted Path that Should Trigger Logout

[REDACTED]

Figure 13 - By not Following the Redirect, the User is Still Authenticated

References

https://cwe.mitre.org/data/definitions/693.html

4 - Sensitive Information Disclosure	
Location/s	[REDACTED]
Remediation Urgency	Later
Risk Level	Medium
CVSS 4.0 Score & Vector	5.3 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N)
CVE/CWE	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Issue Description

The application returns fully unmasked [REDACTED] and [REDACTED] in API responses, despite masking these values in the user interface (UI). This behaviour is consistent across both the mobile and web platforms. While the UI displays only partial information to reduce exposure, any authenticated user can view their complete sensitive data by inspecting network requests. This inconsistent data handling weakens the effectiveness of masking and assumes the client environment is secure, which may not hold true if an attacker gains access to a user's authenticated session or device.

Steps to Reproduce

Web

1. Login to the application while using a HTTP proxy such as Burp Suite to capture background requests.
2. Visit the profile page of the target account.
3. Observe that the account's [REDACTED] is censored.
4. View the background traffic captured in Burp Suite and observe the request to get

the [REDACTED] containing the full unmasked [REDACTED].

Mobile

1. Login to the mobile application while using a HTTP proxy such as Burp Suite to capture background requests.
2. Visit the profile tab and observe that the [REDACTED] are masked.
3. View the background HTTP request inside Burp Suite containing the entire [REDACTED] returned without masking.

Impact

If an attacker were to compromise a user's session or gain unauthorised access to their account, they could easily retrieve full [REDACTED] that the UI was designed to obscure. This increases the risk of downstream abuse such as targeted social engineering, fraud, or identity misuse, particularly if additional protections like session management or device validation are weak or absent.

Remediation

Sensitive information, such as [REDACTED] and [REDACTED], should only be returned in API responses when absolutely necessary. Data masking should be enforced consistently across both the frontend and backend to limit exposure in the event of session hijacking or unauthorised access. Consider applying role-based data access controls and minimising the exposure of sensitive fields unless explicitly required for a specific transaction or workflow.

Associated / Relevant Evidence

[REDACTED]

Figure 14 - Application Profile Page with Masked [REDACTED]

[REDACTED]

Figure 15 - Background API Request Showing Unmasked [REDACTED]

[REDACTED]

Figure 16 - Mobile Application Profile Page Showing Masked [REDACTED]

[REDACTED]

Figure 17 - Background Mobile API Showing the Unmasked [REDACTED]

References
https://cwe.mitre.org/data/definitions/200.html

5 - Account Enumeration Using Application Error Messages	
Location/s	[REDACTED]
Remediation Urgency	Later
Risk Level	Medium
CVSS 4.0 Score & Vector	5.3 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N)
CVE/CWE	CWE-203: Observable Discrepancy
Issue Description	
The application is vulnerable to account enumeration through its API responses. By modifying a user ID parameter in authenticated requests, it is possible to infer the existence of valid accounts based on whether the server returns a generic success or error message. When a valid user ID is supplied, the request completes without error. When an invalid or non-existent user ID is used, the response includes an error indicating that the account does not exist. While no direct user data is disclosed, this discrepancy allows an attacker to	

enumerate valid user IDs through automated probing.

Steps to Reproduce

1. Login to the application using a HTTP proxy such as Burp Suite to capture background requests.
2. Using the endpoint mentioned above (or other endpoints of a similar nature) modify the ID to that of another user account that exists.
3. Observe that the application returns a valid 200 response with JSON.
4. Repeat the steps with a known invalid ID, observe a request rejected response. This difference showcases the other account ID exists.

Impact

This behaviour enables attackers to enumerate valid account identifiers, which can be used as a foundation for targeted attacks such as credential stuffing, phishing, or social engineering. Even without exposing sensitive user data directly, the ability to confirm which accounts exist reduces the anonymity and security of users and may aid in further exploitation if combined with other vulnerabilities.

Remediation

Standardise API error responses to avoid disclosing whether an account exists. Use generic error messages for both valid and invalid user IDs, and implement rate limiting and monitoring to detect and block enumeration attempts. Ensure that authorisation is enforced consistently and that user identifiers are not exposed or guessable.

Associated / Relevant Evidence

[REDACTED]

Figure 18 - API Request Made by an Authenticated User to Their ID

[REDACTED]

Figure 19 - API Request Made After Modifying the ID to Another User ID [REDACTED] Figure 20 - API Request Made After Modifying the ID to an Invalid ID
References
https://cwe.mitre.org/data/definitions/203.html

6 - Unsanitised File Upload	
Location/s	[REDACTED]
Remediation Urgency	Now
Risk Level	Medium
CVSS 4.0 Score & Vector	5.1 (CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:P/VC:N/VI:L/VA:N/SC:N/SI:N/SA:N)
CVE/CWE	CWE-434: Unrestricted Upload of File with Dangerous Type
Issue Description	
The application accepts PDF file uploads without performing sufficient validation or sanitisation of the file content. A test PDF containing an embedded JavaScript-based Cross-Site Scripting (XSS) payload was successfully uploaded and later executed when the file was downloaded and opened. This demonstrates that the platform does not restrict or sanitise active content within supported file types. While the file currently appears accessible only to the uploading user, there is potential for exposure to others if the application allows file	

sharing, download links, or administrative review of uploaded documents.

Steps to Reproduce

1. Login to the application.
2. Visit the account's document section.
3. Upload a document that contains the potentially malicious embedded payload.
4. Download the document after it has been uploaded.
5. Launch the file locally in a browser and see that the XSS payload executes.

Impact

If malicious files can be accessed by other users, this vulnerability may be exploited to execute XSS attacks through trusted downloadable content. Victims opening the file in a browser or supported viewer could unknowingly trigger malicious scripts, leading to session compromise, credential theft, or further exploitation. Even if file access is currently restricted, the lack of sanitisation introduces risk in multi-user environments or future features that involve file sharing or processing.

Remediation

Ensure that uploaded files are strictly validated and sanitised before storage or access. Implement server-side checks to detect and reject active content within files, particularly for formats such as PDF that support embedded scripts. Use content disarm and reconstruction tools where feasible. Restrict access to uploaded content, avoid rendering untrusted files in-browser, and log all upload activity to monitor for abuse.

Associated / Relevant Evidence

[REDACTED]

Figure 21 - Uploading a PDF File with Embedded JavaScript

[REDACTED]

Figure 22 - Downloading the Previously Uploaded File

[REDACTED]

Figure 23 - Launching the File Locally, which Executes the Embedded XSS

References

https://cwe.mitre.org/data/definitions/434.html

7 - No Rate Limit Implemented	
Location/s	[REDACTED]
Remediation Urgency	Later
Risk Level	Informational
CVSS 4.0 Score & Vector	0.0 (Informational Finding)
CVE/CWE	CWE-770: Allocation of Resources Without Limits or Throttling

Issue Description

The application lacks rate limiting controls on several functional endpoints, including those responsible for updating profile information and sending messages. This behaviour was confirmed in both the mobile API and the web application, where repeated requests could be sent in rapid succession without triggering any throttling, delays, or error responses. The absence of rate limiting exposes these endpoints to abuse and suggests that server-side

protections against excessive usage or automation have not been implemented.

Steps to Reproduce

Web

1. Login to the application while proxying the HTTP requests with a tool such as Burp Suite.
2. Visit the profile page.
3. Open the personal information section and click save.
4. Find the sent request in Burp Suite.
5. Using Intruder, send 150 requests within a 5 second period.
6. Observe that all requests are successful without any form of rate limiting.

Mobile

1. Login to the application while proxying the HTTP requests with a tool such as Burp Suite.
2. Visit the messages section and create a new message.
3. Find the background HTTP request sent.
4. Using Intruder, send 150 requests within a 5 second period.
5. Observe that the application does not utilise any rate limit restrictions.

Impact

Without rate limiting, attackers can automate high volumes of requests to abuse functionality such as spamming messages, brute-forcing changes, or manipulating profile data at scale. This can degrade service performance, impact other users, and potentially lead to reputational damage. In cases where these functions tie into external systems or notifications, the impact could extend beyond the application itself and lead to indirect costs or abuse.

Remediation
Implement server-side rate limiting on all endpoints that perform sensitive or state-changing operations. Define thresholds appropriate to each function's expected usage pattern and apply user, IP, or session-based tracking where necessary. Include alerting and logging mechanisms to detect and respond to unusual activity. Rate limiting should be applied consistently across both web and mobile channels.
Associated / Relevant Evidence
[REDACTED] Figure 24 - Saving Personal Information on the Profile Page [REDACTED] Figure 25 - Repeating the Save Request 150 Times within a Few Seconds [REDACTED] Figure 26 - Sending a New Message Inside the Mobile App [REDACTED] Figure 27 - Repeating the New Message 150 Times within a Few Second Period [REDACTED] Figure 28 - Viewing the Sent Messages Inside the App
References
https://cwe.mitre.org/data/definitions/770.html

8 - Unauthenticated Access To Authenticated Functions	
Location/s	[REDACTED]
Remediation Urgency	Later
Risk Level	Informational
CVSS 4.0 Score & Vector	0.0 (Informational Finding)
CVE/CWE	CWE-284: Improper Access Control

Issue Description

The application permits unauthenticated access to search functionality that is intended to be restricted to authenticated users. By directly interacting with the API endpoint responsible for search, an attacker can bypass authentication checks and perform searches without logging in. While the underlying data appears to be publicly available information (such as [REDACTED]), the ability to access features designed for authenticated users without proper authorisation indicates a failure in enforcing access controls. This weakens the application's trust boundaries and could expose other endpoints to similar misuse if not consistently protected. As a note, in some cases in order to perform a search, it will require an ID number that has to be in the correct expected format to work (this would require additional brute forcing to work correctly).

Steps to Reproduce

1. Visit the application authenticated while proxying the HTTP traffic with a tool such as Burp Suite.
2. Using the search bar, type any arbitrary word to search, such as "test".
3. View the background request made to the application in Burp Suite, and remove all

cookies and headers so the request is fully unauthenticated.

4. After sending the request, observe the application performs a search without any errors and does not require authentication.

Impact

Allowing unauthenticated access to functionality intended for authenticated users undermines the principle of least privilege and may encourage further enumeration or abuse. Although the data returned may not be sensitive in this instance, the lack of access controls creates an opportunity for attackers to probe additional endpoints for unintended exposure, which could lead to more significant data leaks or unauthorised operations if similar flaws exist elsewhere.

Remediation

Implement strict server-side access controls to ensure all functionality requiring authentication is consistently enforced across the application. All API endpoints should validate the user's authentication and authorisation status before processing requests, regardless of whether the returned data appears sensitive. Conduct a full access control review to identify and remediate any other endpoints with insufficient protections.

Associated / Relevant Evidence

[REDACTED]

Figure 29 - Performing a Search as an Authenticated User

[REDACTED]

Figure 30 - Performing a Search After Removing All Forms of Authentication from the Request

References

https://cwe.mitre.org/data/definitions/284.html

9 - Insecure Content Security Policy Header	
Location/s	[REDACTED]
Remediation Urgency	Later
Risk Level	Informational
CVSS 4.0 Score & Vector	0.0 (Informational Finding)
CVE/CWE	CWE-693: Protection Mechanism Failure

Issue Description

The web application's CSP includes several insecure directives within its script-src configuration, reducing its effectiveness as a defence against XSS and similar attacks. Specifically, the policy allows 'unsafe-inline', which permits inline scripts and event handlers to execute, and 'unsafe-eval', which enables the use of dangerous JavaScript functions like eval(). The policy also includes 'self', which can be risky if user-uploaded files, JSONP, or AngularJS libraries are served from the same origin. Additionally, multiple external script sources such as doubleclick.net, js-agent.newrelic.com, and bam.nr-data.net are trusted, which may introduce further exposure if those domains serve JSONP or vulnerable libraries. The current configuration lacks modern safeguards like strict-dynamic and CSP nonces or hashes.

Steps to Reproduce

1. Visit the application in the browser.
2. Open the browser development tools, use the network tab, and refresh the page.
3. Observe the application's Content-Security-Policy.

Impact
The permissive CSP increases the attack surface for XSS and other script-based attacks by allowing execution of potentially untrusted scripts. If an attacker is able to inject malicious content into the application, the current CSP would do little to prevent script execution. This could lead to session hijacking, data theft, or other client-side compromise scenarios, particularly if additional vulnerabilities are present that enable content injection.
Remediation
Revise the CSP to eliminate the use of 'unsafe-inline' and 'unsafe-eval', and consider adopting strict-dynamic with properly implemented nonces or hashes to ensure only trusted scripts are executed. Review all external script sources to ensure they are strictly necessary and that they do not serve dynamic JavaScript, JSONP, or AngularJS libraries. Remove 'self' unless it is confirmed that no user-uploaded or dynamic scripts are served from the same origin. Apply a least-privilege approach to all CSP directives and test thoroughly before deployment.
Associated / Relevant Evidence
[REDACTED] Figure 31 - Showing Application Headers with the Browser Development Tools [REDACTED] Figure 32 - Using Google CSP Evaluator to Identify Misconfigurations
References
https://cwe.mitre.org/data/definitions/693.html

5. Walkthrough

The assessment began by mapping out the functionality and structure of both the web and mobile applications. Initial attention was placed on authentication flows and session management behaviour, where OSec observed that a session token was issued immediately after successful username and password authentication, before MFA was completed. By inspecting the issued token and testing authenticated endpoints, OSec confirmed that full access could be obtained without ever passing the MFA challenge.

With authenticated access established, testing moved to reviewing API behaviour and access control enforcement. While interacting with various endpoints, OSec discovered that modifying user-supplied identifiers in requests allowed access to other users' data, including [REDACTED] and [REDACTED]. This behaviour was consistent across both platforms and led to the identification of several IDORs.

During functional testing of features such as messaging and profile updates, it became clear that no rate limiting was in place. Repeating requests rapidly showed no throttling or blocking behaviour, confirming that these endpoints could be abused at scale. In parallel, the team observed that sensitive data such as [REDACTED] and [REDACTED] were returned in full within API responses, even though masked in the user interface. This discrepancy was uncovered by inspecting the raw JSON responses in the browser and via proxy tools.

File upload functionality was then tested, where PDF documents containing embedded JavaScript were uploaded without rejection. When downloaded and opened, these files triggered JavaScript execution, confirming that no sanitisation or restriction on active content was in place. Additionally, the CSP header was reviewed and found to contain permissive directives such as 'unsafe-inline' and 'unsafe-eval', weakening browser-based protections.

Lastly, the application’s built-in session protection, which triggers a logout when certain suspicious behaviour is detected, was tested. OSec found that by preventing the client-side redirect that triggers the logout, the session remained active, and the user could continue interacting with the application, effectively bypassing the protective mechanism.

6. Conclusion

Testing of the [REDACTED] application environment identified several vulnerabilities that, when considered together, represent moderate risk to the application and its users. These findings point to broader issues around access control enforcement, session handling, and data exposure that, if left unaddressed, could be leveraged by an attacker to compromise user privacy or abuse platform functionality.

One of the most serious concerns identified was the ability to bypass MFA by using an authorisation token before the second factor had been completed. This undermines the protection MFA is meant to provide and could allow full account access if user credentials were ever compromised. In tandem with widespread IDOR vulnerabilities, it would be possible for an attacker to view other users' account balances and personal information simply by manipulating identifiers in requests.

The lack of rate limiting across critical features such as [REDACTED] and [REDACTED] allows these issues to be exploited at scale. In addition, sensitive information, such as [REDACTED] and [REDACTED], was returned in API responses, even though the interface masked them from view. This inconsistency increases the risk of sensitive data exposure if a session token is stolen or misused.

The assessment also uncovered client-side risks including a CSP configuration which permits relaxed directives, weakening browser protections. Combined with a file upload feature that allows embedded JavaScript in PDF files, there is potential for an attacker to deliver malicious content to other users if files are ever shared or reviewed by higher-privileged roles.

To address these concerns, OSec recommends improving the enforcement of session and object-level access controls, ensuring that no user gains access to protected functionality without fully completing MFA. API responses should be reviewed to minimise the exposure of sensitive information, and rate limiting should be applied to all functional endpoints. On the client side, the CSP should be hardened, and file uploads should be scanned or restricted to prevent active scripting.

While none of the findings resulted in a full breach during testing, the combination of access control issues, session weaknesses, and data exposure create realistic attack paths that could be exploited by a determined threat actor. [REDACTED] should address these areas to reduce risk and strengthen the environment's overall security posture. Regular assessments and proactive security improvements are recommended as the application continues to evolve.

7. Appendix A - Identified IDOR Locations

Platform	Endpoints
Web Application	[REDACTED]
Mobile Applications	[REDACTED]