



Mobile App / API Penetration Test

Prepared for: Sample Inc

COMPLETE REPORT

April 16, 2026

Red Sentry

This engagement was performed in accordance with the Statement of Work, and the procedures were limited to those described in that agreement. The findings and recommendations in this report reflect the systems, scope, and timeframe defined in the Statement of Work.

While every effort was made to provide thorough coverage, no security assessment can identify every possible issue, and results should not be interpreted as a guarantee of complete security.

Contact Information

+1 (888) 337-0467

hello@redsentry.com



TABLE OF CONTENTS

Summary.....4

 Project Overview..... 4

 Goals..... 4

 Dates..... 4

 Findings..... 4

 Scope..... 5

Tests Performed.....6

Findings.....7

Appendices.....9

SUMMARY

Project Overview

Sample Company engaged Red Sentry to assess the security of their mobile application. The following report details the findings identified during the course of the engagement, which started on January 1st, 2026.

Goals

Identify vulnerabilities in the company’s mobile applications following the OWASP Top 10 Mobile methodology.

Dates

April 08, 2026	April 08, 2026 - April 17, 2026	April 16, 2026
KICKOFF	TESTING PERIOD	DELIVERY

Findings Overview

During the assessment, the testing team identified multiple security findings across the environment.

A finding represents a unique security issue with a distinct root cause.

Instances represent the number of instances where that issue appears within the assessed environment.

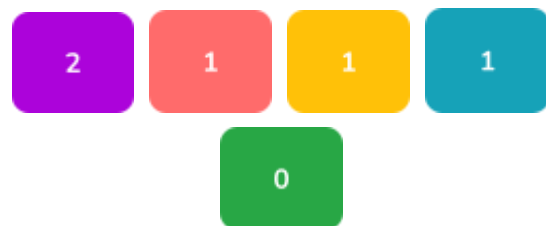
The following table summarizes the identified findings and the number of Instances observed during testing.

Severity	Finding ID	Instances
CRITICAL	TZPDX6V-F0001	1
	TZPDX6V-F0002	1
HIGH	TZPDX6V-F0003	1
MEDIUM	TZPDX6V-F0004	1
LOW	TZPDX6V-F0005	1

Overall Grade



GRADE



Scope

- Sample Company Android Application
- Sample Company IOS Application

EXECUTIVE SUMMARY

The assessment team conducted a penetration test on the mobile application. During the engagement, a total of 5 exploits were found: 2 critical, 1 high, 1 medium, and 1 low severity finding. These vulnerabilities include Pre-Account Takeover, Arbitrary Account Takeover, Weak JWT HMAC Secret, and Password Confirmation Bypass. Each of these exploits poses a distinct threat to the application's security framework and user privacy.

The Pre-Account Takeover vulnerability stems from the lack of email verification in the account registration process. Attackers can exploit this flaw by registering an account with a victim's email address. When the victim later uses Google OAuth, the application automatically links the legitimate login to the attacker's pre-registered account. The assessment team's tests confirmed this vulnerability, demonstrating that an attacker could gain full access to a user's account without owning the associated email address.

Another critical issue is the Arbitrary Account Takeover vulnerability, identified in the application's password reset process, which lacks rate limiting. This flaw allows attackers to brute force the password reset PIN, enabling them to reset and take over any user account.

The Weak JWT HMAC Secret vulnerability was discovered in the application's use of a weak secret key for signing JSON Web Tokens (JWTs). The team was able to crack the key ("supersecret"), enabling attackers to forge JWTs and potentially gain unauthorized access to user accounts. While additional checks limited the team's ability to fully exploit this vulnerability, an attacker could bypass these checks with enough time.

The Password Confirmation Bypass vulnerability was identified in the password change process. The application fails to perform server-side verification of the user's current password, allowing attackers to bypass the need to know the current password and directly interact with the server to change a user's password. This vulnerability exposes user accounts to unauthorized changes and potential account takeovers.

To mitigate these vulnerabilities, the testing team recommends implementing email verification, introducing rate limiting and account lockout mechanisms, using strong, randomly generated secrets for JWTs, enhancing API security, and ensuring server-side verification of the current password during password changes. Addressing these vulnerabilities is essential for safeguarding the application against unauthorized account access and maintaining the integrity and privacy of user data.

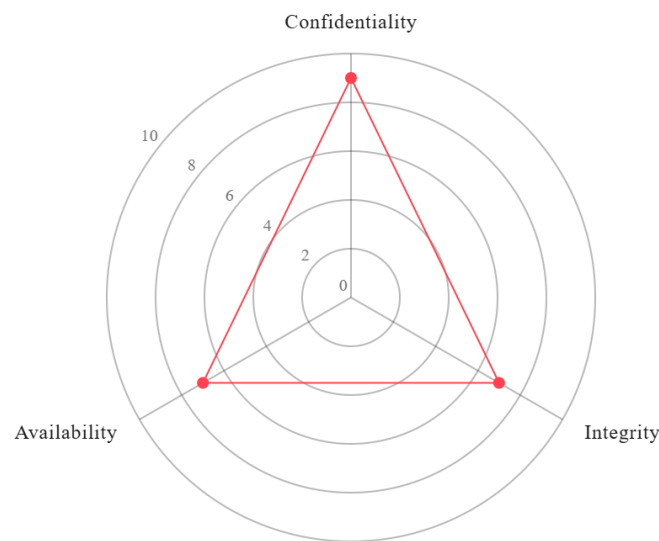


Figure 1: Security risk chart

As the image above shows, there's an overall high risk, as most of the security elements' risk scores are equal to or above 7.

An overall risk below 4 shows low business impact, between 4 and 7 shows medium impact, and above 7 shows high impact.

Test Performed

In Mobile Application Engagement, the main goal is to discover vulnerabilities from a Black Box and a White Box perspective using the OWASP Mobile top 10 as a guide. This includes the following tests:

Improper Credential Usage

This refers to issues where credentials (such as passwords, tokens, and keys) are used improperly, leading to vulnerabilities like credential stuffing and password reuse.

Inadequate Privacy Controls

This involves vulnerabilities where privacy-related data is mishandled or insufficiently protected. Privacy controls are concerned with protecting Personally Identifiable Information (PII), e.g., names, addresses, credit card information, etc.

Inadequate Supply Chain Security

This involves vulnerabilities related to third-party components or dependencies in the software supply chain, such as insecure libraries, outdated components, and CVEs related to the technology stack in use.

Insecure Authentication/Authorization

This category covers vulnerabilities related to authentication and authorization mechanisms, such as weak authentication methods, improper authorization checks, or insufficient access controls. It also covers attempts to spot weaknesses in the way session cookies or tokens are implemented.

Insecure Communication

This refers to vulnerabilities related to the insecure transmission of data over networks, such as a lack of encryption, improper handling of sensitive data in transit, or the use of weak protocols.

Insecure Data Storage

These vulnerabilities involve improper storage of sensitive data, such as storing passwords in plain text, inadequate encryption of stored data, or improper handling of files containing sensitive information.

Insufficient Binary Protections

These vulnerabilities concern weaknesses in protecting an application's binary or executable code, including issues like a lack of anti-debugging measures and weak obfuscation.

Insufficient Cryptography

This involves vulnerabilities due to the use of outdated or insecure cryptographic algorithms, improper management of encryption keys, or weak encryption methods.

Insufficient Input/Output Validation

These vulnerabilities occur when an application fails to properly validate or sanitize input received from users or external systems, which can lead to issues like injection attacks (e.g., SQL injection, XSS) or command injection.

Security Misconfiguration

Security misconfiguration in mobile apps refers to the improper configuration of security settings, permissions, and controls that can lead to vulnerabilities and unauthorized access

Mobile App / API Penetration Test Findings

PRE ACCOUNT TAKEOVER

CRITICAL

Identifier	TZPDX6V-F0001
Severity	Critical
Instances	1

Description

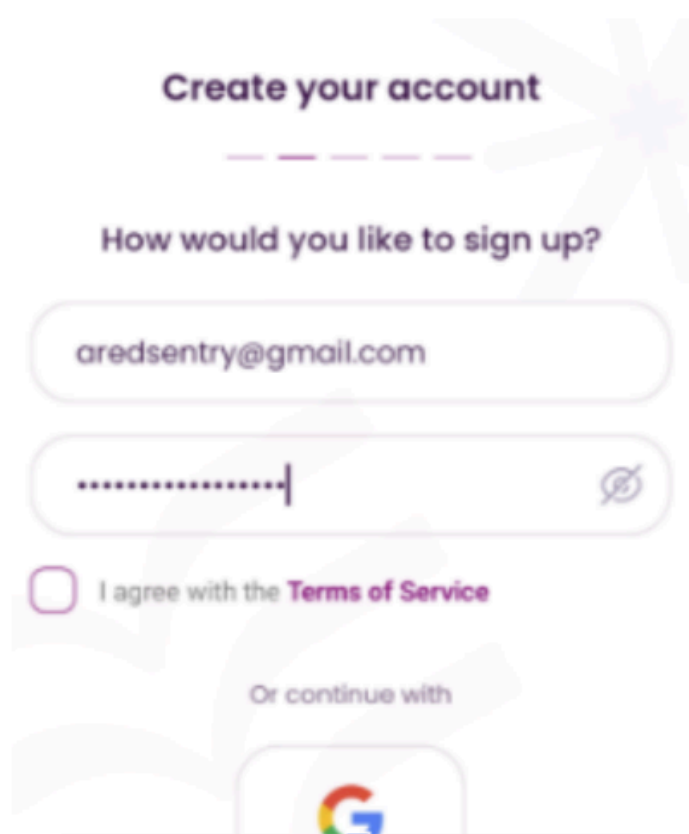
In the Pre-Account Takeover exploit, an attacker registers an account with a victim's email address in an application that lacks email verification. This anticipatory action allows the attacker to gain control of the victim's account when the victim later attempts to sign in using a third-party authentication service such as OAuth as the application may link the victim's legitimate login to the attacker's pre-registered account.

Instances (1)

- api3.SampleCompany.com

Evidence

While examining the mobile application, the assessment team discovered that the application has two ways of logging into the platform. You can log in using your username and password, or you can use Google OAuth.



The screenshot shows a mobile application interface for creating an account. The title is "Create your account" in a bold, dark font. Below it is a dashed line. The question "How would you like to sign up?" is displayed. There are two input fields: the first contains the email address "aredsentry@gmail.com", and the second is a password field with dots and a visibility icon. Below the password field is a checkbox labeled "I agree with the Terms of Service". At the bottom, there is a link "Or continue with" followed by a Google logo.

Figure 2: Create Account Form

As shown in the image above the team created an account with the email “aredsentry@gmail.com”. However, when creating the account, the platform does not verify the user owns the email address that was submitted; the platform only verifies the person's phone number. This means attackers can create accounts with emails they don't own.

Next, the team logged in to the application using the email / password that was created to verify the account had been made successfully. The team then logged into the application using the Google OAuth method for “aredsentry@gmail.com”. After logging into the platform, the team discovered that the user's OAuth account was linked to the account created earlier.

When we created the first account, we didn't have to verify the user's email; this means we can gain access to an account without owning the email address associated with that account. Next, we logged into the platform using OAuth. Since the platform already had an account with that email, it linked the OAuth login to the account the team already created.

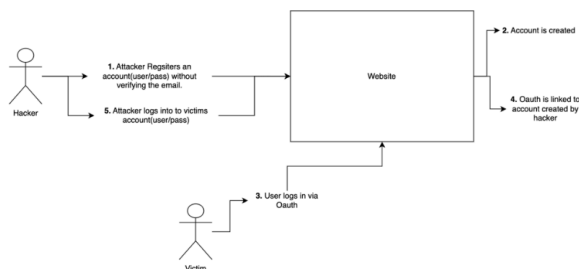


Figure 3: Pre Account Takeover Example

As shown in the screenshot above, an attacker could register accounts with emails they don't own. This is because the application does not perform email validation when creating an account. If a legitimate user were to then log in to the application via OAuth, their account would be linked to the attacker's account. This would allow the attacker full access to the user's account since the attacker can log in via the email/password that they set up.

Steps for Recreation

1. Open the app and go to **Sign Up**.
2. Register using an email you do **not own** (e.g., victim@example.com) and your phone number.
3. Note that **no email verification** is required.
4. Log in using the created email/password.
5. Log out and choose **Login with Google (OAuth)**.
6. Authenticate with the same email via Google.
7. Observe the OAuth account is **automatically linked** to the existing account.
8. Confirm attacker retains access via email/password.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Enforce Email Verification: Require email verification for all new account registrations. This ensures that the email address is controlled by the person attempting to register, preventing attackers from registering accounts with someone else's email.
- Link Accounts Carefully: When a user tries to sign in with a third-party service (like Google or Facebook), check if the email associated with that service already exists in your system. If it does, but the account was not previously linked to the third-party service, prompt the user for additional verification before linking the accounts.

Resources

Oauth Misconfiguration

<https://blog.securelayer7.net/oauth2-0-misconfiguration/>

Hacker Tricks Blog

<https://book.hacktricks.xyz/pentesting-web/oauth-to-account-takeover#ebe4>

Hackerone Report

<https://hackerone.com/reports/1074047>

ARBITRARY ACCOUNT TAKEOVER

CRITICAL

Identifier	TZPDX6V-F0002
Severity	Critical
Instances	1

Description

Arbitrary Account Takeover refers to a critical security vulnerability that allows an attacker to gain unauthorized control over any user account within an application. This type of vulnerability arises when there are flaws or weaknesses in the account authentication or recovery processes, such as inadequate protection against brute force attacks or other exploitation methods. The vulnerability enables an attacker to bypass normal authentication procedures, allowing them to access and control user accounts without legitimate credentials. This can lead to significant breaches of privacy, data theft, and unauthorized actions performed under the compromised accounts.

Instances (1)

- api3.SampleCompany.com/api/v1/auth/verifyUserForgotPasswordOtp

Evidence

While examining the application, the team noticed there was a “forgot password” button. Upon clicking that button, we inputted an email address, which was sent a 6-digit PIN. The application would use this PIN to verify that the user owned the email address to which they requested a password change.

A screenshot of a web form titled "Forgot password" in a bold, dark blue font. Below the title, the text "Enter the 6-digit code we sent to aredsentry@gmail.com" is displayed in a smaller, dark blue font. Underneath this text is a row of six empty circular input fields for a 6-digit PIN. Below the input fields is a link labeled "Resend Code" in a blue, underlined font. At the bottom of the form is a large, rounded rectangular button with the text "Submit" in a dark blue font. The entire form is set against a light gray background.

Figure 4: Forgot password PIN

After reviewing this functionality, the team discovered that the application did not perform any rate limiting when guessing the password reset PIN sent to the user's email address. However, the PIN did seem to expire after about 10 minutes, but due to the missing rate limiting when submitting PINs, the team was able to brute force the reset PIN.

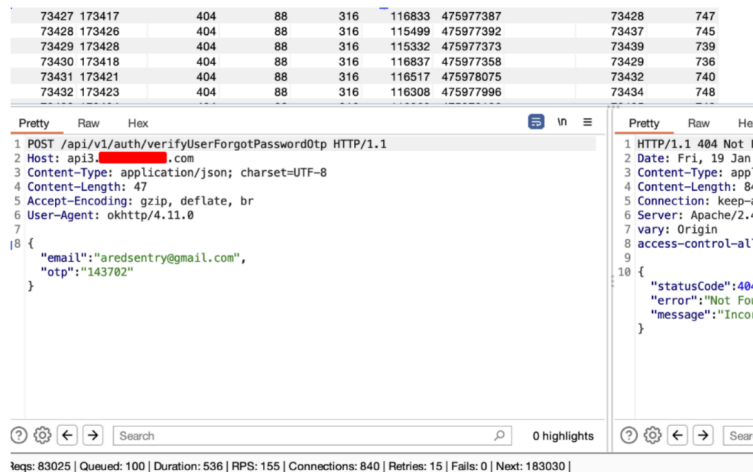


Figure 5: PIN Bruteforce in Burp Suite using Turbo Intruder

As shown in the image above, the team was able to send thousands of PIN requests without being blocked or rate-limited. After a few minutes, the team was able to cycle through enough guesses to find the user's password reset PIN.

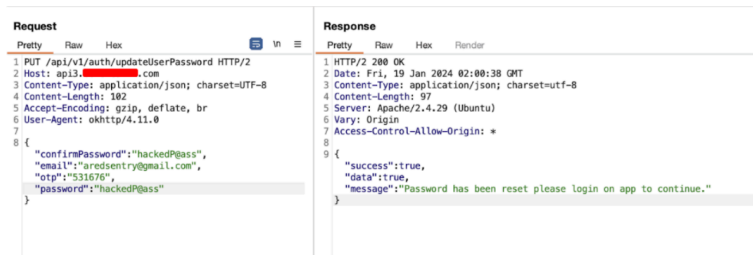


Figure 6: Password change with PIN

As shown in the image above, once the team discovered the password reset PIN, we were able to change the account password successfully. An attacker could leverage this exploit to change any user's password by brute-forcing the password reset PIN.

Steps for Recreation

1. Go to **Forgot Password** on the login page.
2. Enter a target user's email address.
3. Capture the request used to submit the 6-digit reset PIN.
4. Use an automated tool (e.g., Burp Suite) to **brute-force the PIN**.
5. Observe **no rate limiting or lockout** during repeated attempts.
6. Successfully guess the valid PIN within its expiry time (~10 minutes).
7. Use the valid PIN to **reset the account password**.
8. Log in with the new password → **account takeover achieved**.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- **Implement Rate Limiting:** Introduce a rate limit on the number of attempts allowed to enter the password reset PIN. This will prevent attackers from making unlimited guesses and significantly reduce the risk of brute-force attacks.
- **Account Lockout Mechanism:** Implement an account lockout mechanism that temporarily disables the password reset function after a certain number of failed attempts. This will further deter brute force attacks.
- **PIN Complexity and Randomness:** Increase the complexity and randomness of the password reset PIN. Consider using a longer PIN or a combination of letters and numbers to make it more difficult to guess.
- **Multi-Factor Authentication (MFA):** Implement multi-factor authentication for password resets. This could involve sending a verification code to a user's registered mobile device in addition to the email PIN.

Resources

Account takeover

<http://www.cloudflare.com/learning/access-management/account-takeover/>

Bruteforce protection

<https://auth0.com/docs/secure/attack-protection/brute-force-protection>

WEAK JWT HMAC SECRET

HIGH

Identifier	TZPDX6V-F0003
Severity	High
Instances	1

Description

A Weak JWT HMAC Secret vulnerability refers to a security flaw where the secret key used to sign JSON Web Tokens (JWTs) is insufficiently robust, making it vulnerable to cracking. This weakness typically arises from using a common, easily guessable key that can be found in standard wordlists. The consequence of this vulnerability is that attackers can decipher the JWT secret and generate forged JWTs, which the application mistakenly treats as valid, compromising the authentication process. A secure JWT implementation requires the secret key to be randomly generated, lengthy, and complex, reducing the likelihood of unauthorized access through token forgery.

Instances (1)

- api3.SampleCompany.com

Evidence

While examining the application, the team noticed the mobile app was communicating with a backend API server. To communicate with the backend API, the mobile application includes a JWT token to identify users. When a user signs into the application with their username and password, the application is given a JWT token from the API, which is used as the user's access token.

Request

P Raw Hex JSON Web Tokens JWS

```
1 GET /api/v1/users/homeFeeds/v3?timezone=GMT&timeDay=Night HTTP/2
2 Host: api3. [REDACTED].com
3 Authorization: Bearer
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfawQ10iI2NWE4NzJlNDg4NTl
  iNzc4Y2ViNjllZjU1LCJhY2Nlc3Nub2t1biI6Ikk1bFJ2b3h6VG9sYndSSmpuaVN
  3NWlEN1kwallncmptIiwidXNlc1R5cGU10iJVU0VSIiwiaWF0IjoxNzA1NTM4Mjc
  2LCJleHAiOjE3MDgxMzAyNzZ9.M9C0LB4tGDTt3Q9n-hI0zgSqD5aQ6J_u1AzrHp
  ecq8Y
4 Accept-Encoding: gzip, deflate, br
5 User-Agent: okhttp/4.11.0
```

Figure 7: HTTP request with authorization header

As shown in the above image, the JWT token is located in the “Authorization” header. When a user submits an API call, the backend server uses this header to retrieve the JWT token, which is then used to authenticate the user.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfawQ10iI2NWE4NzJlNDg4NTlNzc4Y2ViNjllZjU1LCJhY2Nlc3Nub2t1biI6Ikk1bFJ2b3h6VG9sYndSSmpuaVN3NWlEN1kwallncmptIiwidXNlc1R5cGU10iJVU0VSIiwiaWF0IjoxNzA1NTM4Mjc2LCJleHAiOjE3MDgxMzAyNzZ9.M9C0LB4tGDTt3Q9n-hI0zgSqD5aQ6J_u1AzrHpecq8Y
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "_id": "65a872e48859b778ceb69ef5",
  "accessToken": "XXV50CtAYH0xHvjNDWjg6ARDIZ856Rtr",
  "userType": "USER",
  "iat": 1705548649,
  "exp": 1708148649
}
```

Figure 8: JWT token decoded

In order to prevent attackers from forging their own JWT tokens, they are signed with a key. When the server gets a JWT token, they use that key to make sure the token has not been tampered with and that it is valid. As shown in the above image, the JWT token contains several pieces of information, such as the user's “_id” which is used to authenticate the user and tie the token to a specific account.

While examining the key, the team was able to crack the key used to sign the application JWT tokens. As shown in the image below, the key used was “supersecret.”

```
Enter JWT  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm9keSI6ImNlZnQ4NTlnZWozY2Y2VWljZjULCjEwIiwiaWF0IjoiMjAxNi02LTIybnRurhk7LarCsEmj13xYwg3M
```

Enter Secret / Key
supersecret

Signature verified

Decoded JWT

```
Headers := {  
  -- "alg": "HS256",  
  -- "typ": "JWT"  
}  
  
Payload := {  
  -- "id": "65a472e48859b778ceb69ef5",  
  -- "accessToken": "HXVSOCATtnvJmjdWjgSARdIZB56nt*",  
  -- "userId": "USER",  
  -- "iat": 1785548649,  
  -- "exp": 1788148649  
}  
  
Signature := "HGoTY--Ex_QMS37QTnrurhk7LarCsEmj13xYwg3M"
```

Figure 9: Cracked JWT HMAC Secret

Knowing this key, the team was able to forge any JWT token we wanted. The team tested the exploit by modifying the EXP value and sending it to the backend API, which resulted in a successful request. The team then tried to change the “_id” field to another user but was unsuccessful as there were additional checks that the team couldn't examine at that time. However, a dedicated attacker with more time might be able to bypass that check, which would allow them to take over any account they want by modifying the token “_id” value.

Steps for Recreation

1. Log in to the application using valid credentials.
2. Capture an API request containing the **JWT token** in the Authorization header.
3. Decode the JWT token to view its contents (e.g., `_id`, `exp`).
4. Attempt to crack the JWT signing key using a tool (e.g., wordlist attack).
5. Identify the weak secret key (e.g., `supersecret`).
6. Forge a new JWT token using the discovered secret.
7. Modify token values (e.g., extend `exp`).
8. Send the forged token in API requests.
9. Observe that the server **accepts the tampered token** → improper JWT validation.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- **Use Strong, Randomly Generated Secrets:** The JWT secret key should be a long, complex, and randomly generated string that is not easily guessable or found in standard wordlists. This increases the difficulty for attackers in cracking the secret.

- **Implement Sufficient Key Length:** Ensure that the secret key is of sufficient length, typically recommended to be at least 256 bits, to provide adequate security against brute-force attacks.
- **Rotate Secrets Regularly:** Periodically rotate the JWT secret key to reduce the risk of exposure over time. Implement a secure process for key rotation that minimizes disruption to users.

Resources

Port Swigger

https://portswigger.net/kb/issues/00200903_jwt-weak-hmac-secret

Intigriti

<https://www.youtube.com/watch?v=ov9yT4WAuzI>

Pentester Academy

<https://blog.pentesteracademy.com/hacking-jwt-tokens-bruteforcing-weak-signing-key-johntheripper-89f0c7e6a87>

PASSWORD CONFIRMATION BYPASS

MEDIUM

Identifier	TZPDX6V-F0004
Severity	Medium
Instances	1

Description

Password Confirmation Bypass is a security vulnerability that occurs when an application's password change process lacks proper server-side verification of the user's existing password. Typically, a user interface may prompt for the current password as part of the reset procedure, but this check is not adequately enforced on the server side. As a result, an attacker can bypass the need to know the current password by directly interacting with the server, effectively circumventing the intended security control. This vulnerability exposes user accounts to unauthorized password changes and potential account takeovers, as it allows the password to be reset without confirming the legitimacy of the requestor's identity.

Instances (1)

- `api3.SampleCompany.com/api/v1/users/resetPassword`

The assessment team noticed the application allows users to change their password by supplying their current password as well as the new password. If the user does not know their current password the application prevents them from changing it. This is a common security measure to prevent attackers from changing users' passwords.

Figure 10: Password change form

[illegible]

Figure 11: Password update via API

As shown in the screenshot above, the team put in the wrong password, but the server still accepted the request and changed the user's password. This means users can bypass the password check on the mobile application by going directly to the API and submitting the request.

This vulnerability exposes user accounts to unauthorized password changes and potential account takeovers, as it allows the password to be reset without confirming the legitimacy of the requestor's identity.

Steps for Recreation

1. Log in to the application.
2. Navigate to the **Change Password** feature.
3. Intercept the password change API request.
4. Modify the request by entering an **incorrect current password**.
5. Send the request directly to the API.
6. Observe the server **accepts the request without validating the current password**.
7. Confirm the password is changed successfully.
8. Log in using the new password → **unauthorized password change possible**.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- **Server-Side Verification:** Ensure that the server-side code verifies the user's current password before allowing a password change. This validation should occur at the API or server level, not just on the client side.
- **Enhance API Security:** Secure the API endpoints handling password changes by implementing checks that verify the authenticity of the current password. Any password change requests that fail this check should be rejected.

Resources

Client Side Validation

https://knowledge-base.secureflag.com/vulnerabilities/inadequate_input_validation/client_side_validation_vulnerability.html

Client Side Validation Bypass



<https://cqr.company/web-vulnerabilities/client-side-validation-bypass/>

USER ENUMERATION

LOW

Identifier	TZPDX6V-F0005
Severity	Low
Instances	1

Description

User enumeration is when a malicious actor can use brute-force techniques to either guess or confirm valid users in a system. The application or service reveals when a username exists either as a consequence of a misconfiguration or as a design decision.

Instances (1)

- `api3.SampleCompany.com/api/v1/auth/forgotPassword`

Evidence

The assessment team discovered a single instance of email enumeration. While reviewing the forgot password functionality, the team noticed the application responded with a verbose error message if an email address was not valid.

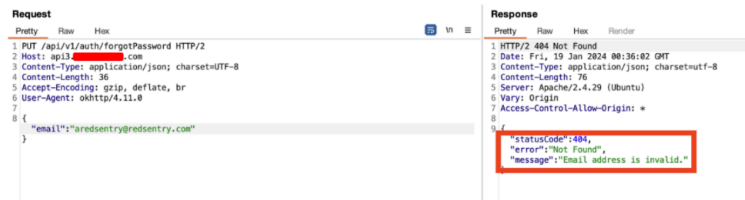


Figure 12: Email enumeration via error message

As shown in the above screenshot, the email “aredetnry@redsentry.com” does not exist. When we typed in “aredsentry@gmail.com” we got another message indicating that the email address was valid.

Attackers could leverage this to build a list of valid email addresses. This list could then be used to launch additional attacks.

Steps for Recreation

1. Navigate to the **Forgot Password** feature.
2. Enter an **invalid/non-existent email address**.
3. Observe the application returns a **specific error message** indicating the email is not registered.
4. Enter a **valid/registered email address**.
5. Observe a **different response/message** indicating the email exists.
6. Compare responses to confirm **email enumeration is possible**.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- An effective remediation would be to have the server respond with a generic message that does not indicate which field is incorrect. If the response does not indicate whether the username or the password is incorrect, the attack surface decreases.

Resources

User enumeration

<https://www.virtuesecurity.com/kb/username-enumeration>

Testing for Account Enumeration and Guessable User Account

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account

APPENDIX (A) SEVERITY DESCRIPTION

The assessment team used the following criteria to rate the findings in this report. Red Sentry derived these risk ratings from the industry and organizations such as OWASP.

The severity of each finding in this report is independent. Finding severity ratings combine direct technical and business impact with the worst-case scenario in an attack chain. The more significant the impact, and the fewer vulnerabilities that must be exploited to achieve that impact, the higher the severity.

Critical

A purple rounded rectangular badge with the word "CRITICAL" in white, uppercase letters.

Vulnerability is an otherwise high-severity issue with additional security implications that could lead to exceptional business impact. Examples: trivial exploit difficulty, business-critical data compromised, bypass of security controls, direct violation of communicated security objectives, and large-scale vulnerability exposure.

High

A red rounded rectangular badge with the word "HIGH" in white, uppercase letters.

Vulnerability may result in direct exposure including, but not limited to: the loss of application control, execution of malicious code, or compromise of underlying host systems. The issue may also create a breach in the confidentiality or integrity of sensitive business data, customer information, and administrative and user accounts. In some instances, this exposure may extend farther in the infrastructure beyond the data and systems associated with the application.

Medium

MEDIUM

Vulnerability does not lead directly to the exposure of critical application functionality, sensitive business and customer data, or application credentials. However, it can be executed multiple times or leveraged in conjunction with another issue to cause direct exposure. Examples include brute-forcing and client-side input validation.

Low

LOW

Vulnerability may result in limited exposure of application control, sensitive business and customer data, or system information. This type of issue provides value only when combined with one or more issues of a higher risk classification. Examples include overly detailed error messages, the disclosure of system versioning information, and minor reliability issues.

Informational

INFO

Finding does not have a direct security impact but represents an opportunity for additional layers of security, is considered a best practice, or has the possibility of turning into an issue over time. Finding is a security-relevant observation that has no direct business impact or exploitability, but may lead to exploitable vulnerabilities. Examples include poor communication between organizations, documentation encouraging poor security practices, or lack of security training.