



MOBILE APP SECURITY AUDIT

SNPIT

Mobile App

CLIENT

SNPIT

PLATFORM

iOS and Android
(Flutter)

LANGUAGE

Dart

REVIEW

August 2026

DOCUMENT

Document information

Project	SNPIT, Mobile App
Client	SNPIT
Platform	iOS and Android (Flutter)
Language	Dart
Repository	snpit-app-develop/ codebase
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Report date	August 2026 (no date in the original; taken from the PDF)

Document history

VERSION	DATE	CHANGE
1.0	August 2026	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	H-01 Location Spoofing Enables Reward Abuse By Intentionally Allowing iOS	8
	L-01 Sentry Crash Reporting Includes User PII Without Proper Scrubbing	9
	I-01 Global iOS App Transport Security Bypass Enabled	10
	I-02 Marketplace One-Time JWT Passed Through URL Query Parameter	11
	I-03 Plaintext Local Storage of Access Tokens and Location Metadata	12
	N-01 Privy Wallet Linking Could Potentially Accept Unverified WebView JavaScript Messages	13
	Disclaimer	15

Executive summary

About SNPIT

SNPIT is a Flutter-based mobile application centered on photo capture, collection, and game-like engagement. Users take and upload photos, join voting battles, complete missions, manage inventory items, use rental and marketplace screens, and access wallet and NFT features.

Summary of the review

Safe Edges reviewed the SNPIT Flutter mobile app codebase (snpit-app-develop). Six issues were identified: one High, one Low, three Informational and one Note. The High issue was that fake-location detection runs only on Android while the backend accepts client-supplied GPS coordinates for location-based rewards and battle multipliers, so spoofed locations can claim rewards. The original report did not record resolution statuses.

Findings by severity



Findings by status

STATUS	ISSUES
● Not recorded	6

Scope

Repository	snpit-app-develop/ codebase
Commit	Not recorded in the original report
Platform	iOS and Android (Flutter)
Language	Dart

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Not recorded in the original report.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
H-01	Location Spoofing Enables Reward Abuse By Intentionally Allowing iOS	High	● Not recorded
L-01	Sentry Crash Reporting Includes User PII Without Proper Scrubbing	Low	● Not recorded
I-01	Global iOS App Transport Security Bypass Enabled	Informational	● Not recorded
I-02	Marketplace One-Time JWT Passed Through URL Query Parameter	Informational	● Not recorded
I-03	Plaintext Local Storage of Access Tokens and Location Metadata	Informational	● Not recorded
N-01	Privy Wallet Linking Could Potentially Accept Unverified WebView JavaScript Messages	Informational	● Not recorded

Detailed findings

H-01 Location Spoofing Enables Reward Abuse By Intentionally Allowing iOS

Severity **High** Status ● Not recorded

DESCRIPTION

The app performs fake-location detection only on Android before camera initialization, with no equivalent check on iOS. The app also does not perform root/device-integrity detection, so rooted or tampered devices are not blocked before submitting capture metadata. The photo upload flow sends client-provided latitude and longitude values as optional request parameters. Backend verification confirms that these coordinates are accepted without server-side location validation.

When a user uploads a photo, the backend stores the submitted latitude and longitude and later geocodes them into `photo_address`. This derived location is used in reward logic, including SnapOnLocation STP bonuses and battle reward multipliers.

Because the backend treats client-provided GPS coordinates as authoritative, an attacker using a spoofed location, rooted/tampered device, or modified client can submit false coordinates. This allows location-based rewards and battle multipliers to be claimed without being physically present at the claimed location.

AFFECTED CODE

`lib/app/camera_snap/components/snap/initialize/camera_initialize_mixin.dart`

```
if (Platform.isAndroid) {
  final isFakeLocation = await DetectFakeLocation().detectFakeLocation();
  if (isFakeLocation) {
    setState(() {
      errorMessage = l10n.homeLocationDisguiseError;
      errorDescription = l10n.homeLocationDisguiseErrorDesc;
      isLoading = false;
    });
    return;
  }
}
// iOS skips fake-location check
```

IMPACT

Attackers can use spoofed locations, rooted/tampered devices, or modified clients to submit false GPS coordinates and claim location-based rewards or battle multipliers without being physically present, enabling reward fraud and unfair gameplay advantages.

RECOMMENDED MITIGATION

Enforce server-side validation for submitted GPS coordinates using impossible-travel checks, rate limits, location-history consistency, and anomaly scoring. Add iOS fake-location detection and

platform-specific device-integrity checks for both Android and iOS. Treat failed integrity checks as high-risk for reward-bearing actions.

L-01 Sentry Crash Reporting Includes User PII Without Proper Scrubbing

Severity **Low** Status ● Not recorded

DESCRIPTION

The app configures Sentry crash reporting with minimal event filtering and sets the Sentry user scope to include personally identifiable information: user UUID, email address, and display name. When the app crashes or encounters an error, this user-identifying data is sent to Sentry's servers along with crash telemetry. The `beforeSend` callback exists for filtering but the callback only drops `PathNotFoundException` events and does not scrub PII.

This exposes user email addresses and display names to Sentry storage and to project members with access to crash reports, increasing privacy and data-minimization risk.

AFFECTED CODE

lib/utils/sentry.dart

```
await SentryFlutter.init((options) {  
  options  
    ..dsn = dsn  
    ..environment = Config.buildTarget  
    ..beforeSend = _beforeSend  
    ..debug = false;  
}, appRunner: () => runApp());
```

lib/app/user/providers/user_provider.dart

```
Sentry.configureScope((scope) {  
  scope.setUser(  
    SentryUser(  
      id: state.uuid,  
      email: state.email,  
      username: state.displayName,  
    ),  
  );  
});
```

IMPACT

User email addresses may be sent to and stored in Sentry as part of crash reports, increasing the risk of unnecessary PII exposure.

RECOMMENDED MITIGATION

Remove email addresses from Sentry user scope. Use only internal Uuid and display name. Add PII scrubbing in `beforeSend` callback to remove emails from stack traces and request data.

I-01 Global iOS App Transport Security Bypass Enabled

Severity Informational Status ● Not recorded

DESCRIPTION

The iOS app sets `NSAllowsArbitraryLoads` to `true` in `Info.plist`, globally disabling App Transport Security restrictions. ATS is iOS's built-in transport security control that requires secure HTTPS connections and enforces minimum TLS standards for network requests.

With this global exception enabled, the app may load cleartext HTTP URLs or connect to endpoints with weaker transport security than ATS would normally allow. The endpoints in the codebase use HTTPS, so no current exploit path is confirmed.

This configuration may also create App Store review risk, because Apple expects ATS exceptions to be justified and narrowly scoped where possible.

AFFECTED CODE

ios/Runner/Info.plist

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSAllowsArbitraryLoads</key>
  <true/>
</dict>
```

IMPACT

Future HTTP endpoints or misconfigured URLs could potentially load without ATS protection, increasing the risk of cleartext traffic exposure or transport-layer interception.

RECOMMENDED MITIGATION

Remove global `NSAllowsArbitraryLoads`. If an exception is unavoidable, use narrowly scoped `NSExceptionDomains` for specific trusted domains and keep all API and WebView URLs on HTTPS.

I-02 Marketplace One-Time JWT Passed Through URL Query Parameter

Severity **Informational** Status ● Not recorded

DESCRIPTION

The app requests a one-time JWT from the marketplace API and appends it to the marketplace authentication URL as a token query parameter. The URL is then opened in an in-app WebView. Placing bearer-like credentials in URLs is a security anti-pattern because URLs are commonly captured by URL-handling surfaces such as WebView history, server access logs, analytics, crash reports, referrers, or screenshots.

AFFECTED CODE

lib/app_core/constants/asset_urls.dart

```
static String marketplaceAuthUrl(String jwt, String languageCode) {  
    return '$marketplaceBaseUrl/auth?token=$jwt&lang=$languageCode';  
}
```

lib/app/market/utils/marketplace_launcher.dart

```
final jwt = await ref  
    .read(marketplaceRepositoryProvider)  
    .getOneTimeJWT(walletAddress: selectedWalletAddress);  
final marketplaceUrl = AssetUrls.marketplaceAuthUrl(jwt, languageCode);  
final uri = Uri.parse(marketplaceUrl);  
await launchUrl(uri, mode: LaunchMode.inAppWebView);
```

IMPACT

The one-time JWT may be exposed through URL-handling surfaces. If captured before redemption or expiry, it can potentially be used to authenticate a marketplace web session as the victim.

RECOMMENDED MITIGATION

Avoid passing tokens in URL parameters; use a backend session exchange or POST-based redemption flow instead. Make marketplace JWTs short-lived, single-use, and invalidate them immediately after use.

I-03 Plaintext Local Storage of Access Tokens and Location Metadata

Severity Informational Status ● Not recorded

DESCRIPTION

The app stores sensitive local data without using platform secure storage. The API bearer token is persisted through SharedPreferences, and offline capture data is stored under the app Documents directory with JSON metadata containing token ID, timestamp, angle, and precise GPS coordinates. If local app data is extracted through device compromise, malware, physical access, or backup/forensic tooling, an attacker may recover the bearer token or offline location metadata. This is a local data protection hardening issue rather than a remotely exploitable vulnerability.

AFFECTED CODE

lib/infrastructure/local_user_access_token/local_user_access_token.dart

```
class LocalUserAccessToken extends BaseSharedPreferencesService<String> {
  @override
  SharedPreferencesKey get key => SharedPreferencesKey.accessToken;
  @override
  Future<void> set(String value) {
    ref.read(apiClientProvider).setAccessToken(value);
    return super.set(value);
  }
}
```

lib/features/offline/offline_model.dart

```
meta.add({
  'id': id,
  'fileName': fileName,
  'tokenId': tokenId,
  'angle': angle,
  'latitude': latitude,
  'longitude': longitude,
  'takenAt': DateTime.now().toIso8601String(),
});
final base = await getApplicationDocumentsDirectory();
final dir = Directory(p.join(base.path, _dirName));
```

IMPACT

If local app storage is extracted, access tokens and precise location metadata may be exposed, enabling API authentication as the victim until token expiry/revocation or revealing the users capture locations.

RECOMMENDED MITIGATION

Store bearer tokens in iOS Keychain or Android Keystore-backed secure storage. Encrypt sensitive offline metadata where feasible and clear sensitive offline state on logout and account deletion.

N-01 Privy Wallet Linking Could Potentially Accept Unverified WebView JavaScript Messages

Severity **Informational** Status ● Not recorded Category Note

DESCRIPTION

The Privy wallet-linking flow opens a remote URL in `CommonWebView`, registers a JavaScript channel named `FlutterPrivyLoginEventsListener`, and treats the received message as a wallet address without client-side origin, schema, or format validation. The message is passed to `integratePrivy(address: address)`, which submits the address to the backend for wallet integration.

The vulnerable component is the client-side trust boundary: remote `WebView` content controls the wallet address submitted to the backend. If an attacker can influence the loaded `WebView` content or navigation target, they could cause the app to submit an attacker-chosen wallet address. However, the current codebase does not prove that an external attacker can control the production Privy page.

AFFECTED CODE

`lib/app_core/ui_parts/common_web_view.dart`

```
await _controller.setJavaScriptMode(JavascriptMode.unrestricted);
await _controller.addJavaScriptChannel(
  entry.key,
  onMessageReceived: (message) {
    entry.value(message.message);
  },
);
```

`lib/app/wallet/privy_connect_page.dart`

```
CommonWebView(
  url: url,
  javascriptChannelActions: {
    _javascriptChannelName: (message) {
      log('onJavaScriptMessage: $message');
      if (context.mounted) {
        context.pop(message);
      }
    },
  },
),
```

`lib/app/wallet/components/snplit_wallet_card.dart`

```
final address = await PrivyConnectPage.show(context);
if (address == null) {
  return;
}
await ref
  .read(externalWalletRepositoryProvider)
  .integratePrivy(address: address);
```

lib/repository/external_wallet/external_wallet_repository.dart

```
Future<void> integratePrivy({required String address}) async {  
    final response = await _externalWalletApi  
        .externalWalletControllerIntegratePrivyWallet(address: address);  
}
```

IMPACT

An unverified wallet address can be linked to the authenticated user account. If the WebView message or client request is influenced, the backend will accept the submitted Privy address, allowing incorrect or unauthorized wallet binding.

RECOMMENDED MITIGATION

Require backend-side wallet ownership verification before linking, such as a signed nonce challenge. Apply strict address format validation and duplicate checks as secondary controls.

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS