Wahoo Eats (Android) – Security Findings

Documented By: Anonymous

8/31/2025

Methodology and Limitations

- Analysis was performed on a decompiled `.apk` file obtained from the Android client.
- Not all source code was reviewed; deeper inspection of smali/disassembled code is pending and may reveal additional issues.
- Testing was conducted from the client perspective only. No access to backend services, infrastructure, or server-side configuration was available. Network-layer and infrastructure-level findings are therefore outside the scope of this review.
- This report was prepared by a single analyst with AI-assisted tooling to accelerate review. While care was taken to ensure accuracy, some risks may be overstated or understated without corroborating backend/system context.

As such, findings should be interpreted as indicative of significant client-side risks rather than a comprehensive security assessment. Verification and further investigation are recommended.

Scope

Source: `AndroidManifest.xml` (package: `com.nextepsystems.wahoo`).

Findings: `AndroidManifest.xml`

Critical

C1. Cleartext Traffic Enabled Globally

Evidence:

```
<application ... android:usesCleartextTraffic="true">
```

Risk: Allows HTTP/plaintext to any host. Enables MITM interception/modification of credentials, student identifiers, and payment/session data. Violates standard transport encryption requirements; elevates FERPA/PCI exposure.

Remediation:

- Set `android:usesCleartextTraffic="false"`.
- Define strict `res/xml/network_security_config.xml` (cleartext disallowed; restrict trust anchors; consider certificate pinning for API domains).
- Ensure all endpoints use `HTTPS`.

C2. Custom-Scheme Deep Link (Hijackable)

Evidence:

```
<activity android:name="com.mobileordering.MainActivity" android:exported="true" ...>
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="wahooeatsbynextep"/>
  </intent-filter>
</activity>
```

Risk: Any third-party app can register the same scheme to intercept callbacks (auth/payment tokens, session links) or trigger unintended app states (intent injection).
Remediation:

- Migrate sensitive callbacks to Verified Android App Links over `https://` with `android:autoVerify="true"` and `assetlinks.json`.
- If scheme retained for legacy use, ensure no secrets/tokens are delivered via scheme URIs; validate intents strictly (host/path, signature checks).

High

H1. Legacy/Broad External Storage Access

Evidence:

```
<application ... android:requestLegacyExternalStorage="true">
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Risk: On affected Android versions, grants broad external storage access. If PII/tokens/logs are written externally, other apps can read or tamper. Increases data-at-rest exposure.
Remediation:

- Remove legacy flag; adopt Scoped Storage (internal app dirs, SAF/MediaStore as needed).
- Drop `WRITE_EXTERNAL_STORAGE`; avoid storing secrets/PII outside internal storage.

H2. No Network Security Config Present

Evidence: No `android:networkSecurityConfig` declared; no `res/xml/network_security_config.xml` found. **Risk:** With cleartext enabled globally, absence of domain-scoped restrictions permits HTTP to any host; no pinning or per-domain policy.
Remediation:

- Add `@xml/network_security_config`; disable cleartext; optionally pin API domains; restrict trust anchors to system CAs only.

Medium

M1. Over-Privileged Location/Bluetooth Set

Evidence:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
                 android:usesPermissionFlags="neverForLocation"/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
```

Risk: Expanded device data access surface for a dining app; privacy impact and potential unintended data collection. Runtime mis-scopes can leak location-derived data.
Remediation:

- Apply least-privilege: remove unused permissions; request at runtime only where essential; document necessity.

M2. Deprecated HTTP Stack Reference

Evidence:

```
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
```

Risk: Indicates potential use of deprecated networking APIs; when combined with cleartext, increases likelihood of weak transport handling.
Remediation:

- Remove legacy library; standardize on modern clients (e.g., OkHttp) with TLS.

M3. Advertising/AdServices in Campus App

Evidence:

```
<uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
<uses-permission android:name="android.permission.ACCESS_ADSERVICES_ATTRIBUTION"/>
<uses-permission android:name="android.permission.ACCESS_ADSERVICES_AD_ID"/>
```

Risk: Privacy risk and unnecessary data sharing vectors in a mandatory dining/payment app.
Remediation:

- Remove unless strictly required; document purpose; honor platform privacy controls.

Informational / Positive Controls

- `android:allowBackup="false"` set (reduces backup exfiltration risk).
- Key `ContentProvider`s marked `exported="false"`.
- Firebase IID receiver guarded by `com.google.android.c2dm.permission.SEND`.

Recommended Baseline Fix Set

- Enforce TLS: `android:usesCleartextTraffic="false"` + strict `network_security_config`.
- Migrate sensitive deep links to Verified App Links over HTTPS.
- Remove legacy/broad storage access; adopt Scoped Storage; never store secrets/PII externally.
- Reduce permissions to least-privilege; justify any remaining location/Bluetooth/Ad ID usage.
- Remove deprecated HTTP stack; standardize on modern TLS clients.

Findings: `res/values/strings.xml`

High

S1. Exposed Google/Firebase Identifiers

Evidence:

```
<string name="google_api_key">AIzaSyDZ1nhh7fzw-6bjOiaHzC-gMYkcA1y7N9k</string>
<string name="google_crash_reporting_api_key">AIzaSyDZ1nhh7fzw-6bjOiaHzC-gMYkcA1y7N9k<
<string name="google_app_id">1:849770599268:android:cf366c8fe82a9c5e646071</string>
<string name="gcm_defaultSenderId">849770599268</string>
<string name="project_id">nextepnotifications</string>
<string name="google_storage_bucket">nextepnotifications.firebasestorage.app</string>
<string name="default_web_client_id">849770599268-hlcs8h0oebrash3b6dfm33npqbka9i8r.apps
```

Risk: While Firebase âĂIJAPI keysâĂİ are not secrets, public identifiers enable abuse if backend rules are permissive (e.g., unauthenticated reads/writes to Firestore/RTDB/Storage) or OAuth clients are not restricted; can lead to data exfiltration or account misuse when combined with plaintext transport.

Remediation:

- Enforce strict Firebase Security Rules (no public read/write; require authenticated access).
- Restrict OAuth client to package name `com.nextepsystems.wahoo` and release keystore SHA-1/256.
- Rotate identifiers/keys if any server-side trust was erroneously placed in them.

Medium

S2. React Native Developer/Debug Strings Present

Evidence:

```
<string name="catalyst_dev_menu_header">React Native Dev Menu (%1$s)</string>
<string name="catalyst_perf_monitor">Show Perf Monitor</string>
<string name="catalyst_reload">Reload</string>
<string name="catalyst_open_debugger_error">Failed to open debugger...</string>
<string name="catalyst_loading_from_url">Loading from %1$sâĂę</string>
```

Risk: Presence of dev strings implies potential for dev features if not disabled in release (remote debugging, bundle URL override). If enabled, attackers could alter runtime or route traffic to attacker-controlled bundles.

Remediation:

- Ensure `getUseDeveloperSupport()` is **false** in release; remove any runtime toggles or dev-only deep links.
- Verify no production code paths load bundles from arbitrary URLs.

Informational

S3. Payments/Scanning-Related UX Strings

Evidence:

```
<string name="wallet_buy_button_place_holder">Buy with Google</string>
... Scandit / barcode scan strings ...
```

Risk: Indicates payments and scanning features exist; when combined with cleartext transport (see Manifest finding C1), increases sensitivity of any transmitted PII/payment data.
Remediation:

- Ensure all payment and scan-related network calls use HTTPS with strict validation; avoid sending PII in query strings.

Findings: `assets/appcenter-config.json.xml`

S1. Exposed Microsoft App Center Secret

Evidence:

```
appcenter-config.json
{
  "app_secret": "**********************************" // Omitted for privacy
}
```

Risk: Hardcoded App Center secret publicly exposed. Allows attackers to send forged crash/analytics events, polluting telemetry or masking malicious activity. While it does not grant direct App Center console access, it weakens trust in monitoring data and may aid further exploitation if combined with other leaks.
Remediation:

- Remove hardcoded secret from client bundle.
- Store App Center credentials securely (e.g., server-side or in CI/CD secrets).
- Rotate the App Center secret immediately.

Findings: `assets/wallet/js/credit-card-GENIUSCHECKOUT.js`

High

J1. Unsanitized Error Message Injection (DOM XSS risk)

Evidence:

```
var errorMsg = '';
...
errorMsg = 'Error: ' + errorSource + '. Reason: ' + errorReason;
var hcMessage = $("#hpcContainer");
$(".alert-danger").remove();
hcMessage.append('<div class="alert alert-danger" >' + errorMsg + '</div>');
```

Risk: Concatenates untrusted strings (e.g., errorReason) directly into HTML without escaping. If upstream values contain HTML, attacker-controlled content can be injected into the DOM (script or markup execution) within the WebView.
Remediation:

- Encode user-controlled data before insertion (e.g., set text via `.text(...)` or use a safe templating/escaping function).
- Render error content as text nodes, not HTML; whitelist message formats.

Medium

J2. Exposed Third-Party Web API Key (Cayan/Genius)

Evidence:

```
var webApiKey = "****************"; // Omitted for privacy
CayanCheckoutPlus.setWebApiKey(webApiKey);
```

Risk: Publicly embedded key may allow unauthorized use of tokenization APIs (e.g., generating tokens), polluting telemetry, or aiding abuse. Impact depends on provider-side controls and domain restrictions.

Remediation:

- Rotate the key; restrict usage by domain/app as supported by the provider.
- Avoid embedding sensitive credentials client-side; prefer server-mediated initialization when possible.

J3. Remote Script Loaded Without Integrity Controls (Supply Chain)

Evidence:

```
var script = 'https://ecommerce.merchantware.net/v1/CayanCheckoutPlus.js';
$.getScript(script, function (...) { ... });
```

Risk: Relies on remote script delivery without Subresource Integrity (SRI) or pinning. If the third-party host or path is compromised, malicious code executes inside the payment context.

Remediation:

- Prefer bundling a pinned version or use a loader that supports SRI.
- If dynamic loading is required, fetch from a vetted, versioned path under strict TLS; monitor checksum changes.

Low

J4. Weak Client-Side Card Validation

Evidence:

```
jQuery.validator.addMethod("creditcard", function (...) {
  var myRe = /\d{16}/;
  var myArray = myRe.exec(ccnum);
  if (ccnum != myArray) { return false; } else { return true; }
});
jQuery.validator.addMethod("cvvvalidate", function (...) {
  var myRe = /^[0-9]{3,4}$/;
  var myArray = myRe.exec(cvv);
  if (cvv != myArray) { return false; } else { return true; }
});
```

Risk: Accepts only 16-digit PANs (excludes valid lengths like 15), lacks Luhn check, and relies on client-side validation which can be bypassed. Not a direct server-side vulnerability if tokenization and backend validation are correct, but increases error handling surface and potential misuse.

Remediation:

- Implement server-side validation and rely on provider-hosted fields/tokenization.
- If client checks are retained: support valid BIN/length ranges and Luhn; avoid brittle regex-only checks.

J5. Window message handler without origin checks (minor)

Evidence:

```
window.addEventListener("message", function(e) {
  window.ReactNativeWebView.postMessage(JSON.stringify("hi"));
});
```

Risk: Listens to all postMessage events without verifying origin. Current handler does not act on attacker input, but may enable message spam or future misuse if expanded.
Remediation:

- Validate `e.origin` and message schema; ignore unexpected origins and payloads.

Notes (linked to Manifest C1)

- Although the payment script URL is HTTPS, global `usesCleartextTraffic="true"` (C1) increases exposure to mixed-content or downgrade mistakes elsewhere. All payment-related navigation and APIs must be enforced over TLS end-to-end.

Findings: `assets/wallet/js/credit-card-FREEDOMPAYHPC.js`

High

F1. Session Key Extraction from iFrame URL

Evidence:

```
var iFrameHTML = $('#hpcContainer iframe')[0];
var iFrameSrc = iFrameHTML.src;
var srcSplit = iFrameSrc.split('?')[1];
var srcParams = new URLSearchParams(srcSplit);
var sessionKey = srcParams.get("sessionKey");
messageData.sessionKey = sessionKey;
```

Risk: Sensitive payment session key is extracted from the iframe URL query string and re-attached to a `postMessage` payload. If the WebView or bridge is compromised, session hijacking or replay attacks are possible.
Remediation:

- Avoid exposing session keys in URL query strings; use secure storage or backchannel APIs.
- Do not propagate keys to the JS/ReactNative layer unless strictly necessary; prefer server-side session binding.

Medium

F2. Window Message Handler Without Origin Validation

Evidence:

```
window.addEventListener("message", function(e) {
  const messageData = e.data;
  ...
  window.ReactNativeWebView.postMessage(JSON.stringify(messageData));
});
```

Risk: Accepts all `postMessage` events without checking `e.origin` or sender. Attacker-controlled iframes or injected scripts can send arbitrary messages, potentially leaking session keys or triggering unintended flows.

Remediation:

- Validate `e.origin` and expected schema before processing messages.
- Drop or sanitize unrecognized message types; enforce strict contract between app and iframe.

F3. Reliance on Client-Side Form Validation Only

Evidence:

```
$form.validate({
  rules: { hostedPaymentfriendlyName: {required: true} },
  ...
  submitHandler: function(){ return false }
});
```

Risk: Only validates presence of âĂIJfriendly nameâĂİ locally; no evidence of server-side validation. Attackers can bypass client checks entirely.

Remediation:

- Ensure all payment data, including friendly name and session binding, are validated server-side.

Notes (linked to Manifest C1)

- Combined with global cleartext traffic (C1), exposing session keys and payment-related data via WebView bridges amplifies interception risk.