ZERODRIFT

APPLICATION SECURITY AUDIT

CLAIM FREE SOL

Security Assessment Report

WEB APPLICATION · SOLANA

STATUS	FINAL REPORT
DATE	JULY 21, 2026
COMMIT	fbce673875255dbc16acd1451df91587acb28c68

Report Contents

Executive Summary

ZeroDrift performed a point-in-time security assessment of Claim Free SOL over July 8-15, 2026. The assessment was limited to the systems and versions listed in Audit Scope and combined expert manual review, AI-assisted analysis, and targeted static analysis.

The assessment identified **7 reportable findings**. Severity and status details are summarized below.

Assessment at a Glance

Project Name	Claim Free SOL
Repository	https://github.com/ClaimFreeSol/Claim-Free-Sol
Commit	fbce673875255dbc16acd1451df91587acb28c68
Contract Address(es)	N/A - repository scope
Audit Timeline	July 8-15, 2026
Report Date	July 21, 2026
Assessment Approach	Expert Manual Review, AI-assisted Analysis, Targeted Static Analysis

Findings at a Glance

Critical	High	Medium	Low	Info	Gas	**Total**
0	0	5	1	1	0	**7**

Summary of Findings

ID	Description	Status
M-01	Pump multi-stage transaction flow closes a shared PDA before prior claim confirmation	Fixed
M-02	Broadcast-only transaction handling causes false success reporting and opaque partial completion	Fixed
M-03	Dependency Scan	Acknowledged
M-04	Platform fees are not bound to the actual recovered amount and may be paid from existing user balances	Acknowledged
M-05	Priceless tokens bypass high-value hiding and can be burned through Select All	Fixed
L-01	Malicious NFT symbols can cause persistent denial of service	Fixed
I-01	Helius RPC key is exposed in the browser and can be abused for quota theft	Fixed

Protocol Overview

Claim Free SOL is a non-custodial Solana wallet cleanup and rent-recovery platform. It enables users to identify eligible token accounts, close inactive or empty accounts, burn unwanted tokens or NFTs, and reclaim the refundable SOL rent locked in those accounts. Users review and approve all actions through their own Solana wallets. The platform also provides optional post-claim applications, including Coin Flip, Dice, Plinko, and Mines.

Official Sources

- https://claimfreesol.com

Audit Scope

Scope Item	Assessed Version
Repository	https://github.com/ClaimFreeSol/Claim-Free-Sol
Commit	fbce673875255dbc16acd1451df91587acb28c68
Contract or Program Address(es)	N/A - repository scope

Methodology and Limitations

The assessment used expert manual review, AI-assisted analysis, and targeted static analysis. Review activity focused on security properties, trust boundaries, access control, state transitions, asset accounting, and failure modes relevant to the supplied scope.

This was a time-boxed, point-in-time assessment. It did not cover components, deployments, integrations, or changes outside the Audit Scope table. Findings reflect the code and information made available during the stated review period.

Detailed Findings

Findings are grouped by severity. Empty severity groups are omitted for easier scanning.

Medium Severity

[M-01] Pump multi-stage transaction flow closes a shared PDA before prior claim confirmation

Severity	Status
Medium	Fixed

Location:

- `AccountBox.tsx:414-447`
- `usePumpfunCashback.ts:237-367, :369-458`

Description

The USDC claim path calls `sendRawTransaction` without waiting for `confirmed` or `finalized` status. Non-user-rejection failures are also swallowed by **`catch`** and execution continues. The native claim path then appends `closeUserVolumeAccumulator` for the shared bonding / AMM `user_volume_accumulator`. This flow does not use a durable nonce and does not enforce ordering through an explicit state machine, so it cannot rely on browser submission order to guarantee the landing order of two independent transactions. The first transaction may also be accepted by the RPC node and later dropped or expire.

Impact

The USDC claim may fail or remain unconfirmed while the close flow still proceeds. If the on-chain close path does not block closure while stable cashback remains unclaimed, users may lose unclaimed USDC rewards. Even if the on-chain program blocks closure, users will see an opaque failure. The issue applies when both the USDC claim path and the native / close path are active.

Recommendation

1. After `sendRawTransaction`, wait for confirmation using `{ signature, blockhash, lastValidBlockHeight }`.

2. If confirmation fails, stop the subsequent close path immediately and do not continue after `catch`.

Client Response

The USDC claim flow now waits for on-chain `confirmed` status before allowing the SOL or WSOL Pump claim flow to continue. If the USDC claim fails, the subsequent Pump SOL or WSOL stage that may close the shared PDA is skipped, while the normal account-close flow can continue independently.

[M-02] Broadcast-only transaction handling causes false success reporting and opaque partial completion

Severity	Status
Medium	Fixed

Location:

- `AccountBox.tsx:423-447, 551-562`
- `TokensBox.tsx:1082-1097`
- `NFTsBox.tsx:1017-1032`

Description

All active transaction paths continue after `sendRawTransaction` returns a signature and do not check the on-chain `status.err`. The token and NFT burn paths show a success notification immediately after the send loop finishes. The empty-account close path in `AccountBox` also reports success without confirmation. In addition, each batch obtains a single blockhash before construction and reuses it across the whole batch, so later transactions can expire if signing or sending takes too long.

Impact

Users may see a success message even when accounts were not closed or assets were not burned because the transaction was dropped, expired, or failed during execution. In multi-batch scenarios, partial success is not tracked per transaction, leaving no reliable basis for retry or support investigation. For irreversible burn and close operations, this reduces auditability and operational transparency.

Recommendation

1. Confirm each transaction with `confirmTransaction`.
2. Do not mark a transaction as successful unless it is confirmed and `err == null`.
3. Record a signature-to-account or signature-to-mint mapping.
4. Refresh the blockhash for each signing batch.
5. Show explicit succeeded, failed, expired, and unknown states in the UI, and support retrying only failed items.

Client Response

The account close, token burn, and NFT burn transaction flows were changed from broadcast-only `sendRawTransaction` handling to send-and-confirm handling. Transactions are only treated as successful after `confirmTransaction` returns successfully with `err == null`, and each signature batch refreshes the latest blockhash to avoid stale blockhash failures.

[M-03] Dependency Scan

Severity	Status
Medium	Acknowledged

Location:

- `package.json`
- `pnpm-lock.yaml`
- `Web3Provider/index.tsx`

Description

The production dependency scan snapshot from `npm audit --omit=dev` reported 94 advisories: 1 Critical, 15 High, 59 Moderate, and 19 Low.

```
xlsx  *
Severity: high
Prototype Pollution in sheetJS - https://github.com/advisories/GHSA-4r6h-8v6p-xvw6
SheetJS Regular Expression Denial of Service (ReDoS) - https://github.com/advisories/GHSA-5pgg-2g8v-p4x9
No fix available
node_modules/xlsx

94 vulnerabilities (19 low, 59 moderate, 15 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix
```

Figure 1: Dependency audit snapshot

The project installs many wallet adapters through the aggregate `@solana/wallet-adapter-wallets` package, while the application appears to instantiate only Phantom, Solflare, Trust, and Coinbase. Some advisories may be located in unused adapters or Node-only paths and may not be directly exploitable in the browser bundle. However, the project does not provide reachability analysis, exception rationale, or an SBOM to prove which advisories are unreachable in the final client.

Recommendation

1. Replace the aggregate wallet package with per-wallet adapter packages for only Phantom, Solflare, Trust, and Coinbase.
2. Upgrade fixable transitive dependencies.
3. Use bundle analysis and SBOM reachability analysis to identify which vulnerable modules are actually shipped.
4. Add CI controls that block newly introduced High and Critical production advisories.

Client Response

The project acknowledged this finding and deferred broad dependency replacement or dependency splitting for a later remediation cycle.

[M-04] Platform fees are not bound to the actual recovered amount and may be paid from existing user balances

Severity	Status
Medium	Acknowledged

Location:

- `src/assets/hooks/usePumpfunCashback.ts:264-266, 324-331, 344-361, 438-452`
- `src/components/CleanUp/Features/AccountBox.tsx:380-383, 514-539`
- `src/components/CleanUp/Features/TokensBox.tsx:870-876, 1047-1071`
- `src/components/CleanUp/Features/NFTsBox.tsx:797-806, 957-1005`
- `src/assets/constants/index.ts:68-73`

Description

The platform fee is paid through separate `SystemProgram.transfer` or SPL token transfer instructions from the user's balance. It is not atomically split on-chain from the actual newly recovered amount in the same transaction.

- Pump fees are based on previously refreshed snapshots of bonding lamports, WSOL, and USDC, not real-time deltas immediately before signing.
- Pump's official `claim_cashback_v2` is permissionless, so rewards may be claimed by another transaction between refresh and landing.
- If the user already has a WSOL or USDC associated token account, the fixed transfer cannot distinguish cashback proceeds from the user's existing principal. A fee can still be paid even when the actual newly claimed amount is zero.
- Empty-account, token, and NFT flows depend on three fixed constants rather than reading the actual lamports in each account to be closed or computing transaction balance deltas:
  - `ONE_ACCOUNT_RENTAL_FEE`
  - `ONE_NFT_ACCOUNT_RENTAL_FEE`
  - `DEFAULT_SERVE_FEE`

In close-account flows, rent is usually returned before the fee transfer, so the fee commonly comes from the recovered amount. However, stale state, incorrect constants, or Pump race conditions can still cause fees to be paid from the user's existing SOL, WSOL, or USDC.

Impact

This can cause fees to be paid even when the actual claim or burn recovery is smaller than expected or zero. The amount is generally bounded by the nominal 15% model, so the issue is rated Medium. It directly violates the business invariant that users should not lose pre-existing SOL or assets.

Recommendation

1. Use an audited on-chain program that splits fees based on the actual claimed amount, or change the product model to an explicit fixed service fee instead of advertising an atomic 15% recovery fee.
2. If client-side fee transfers are retained, first confirm the actual delta, then show the fee and request a second explicit signature. Do not describe this as an atomic 15% fee.
3. For account-closure flows, read the real lamports in each account and display gross proceeds, fee, network cost, ATA rent, and net proceeds.
4. Enforce a maximum fee and `minimumNetProceeds`, and fail closed when state is inconsistent.

Client Response

The project acknowledged the risk and accepted the current client-side fee transfer model for the short term. The team stated that fully addressing the issue would require a new and separately audited on-chain fee splitter or backend-controlled flow, so the fee architecture was not changed in this remediation round.

[M-05] Priceless tokens bypass high-value hiding and can be burned through Select All

Severity	Status
Medium	Fixed

Location:

- `src/assets/hooks/`
  - File: `useCleanUpFeatures.ts`
  - Variables: `value`, `isHighValueHidden`
- `src/components/CleanUp/Features/`
  - File: `TokensBox.tsx`
  - Function: `handleCloseTokenAccounts`

Description

The "hide high-value tokens" guard only marks a token as `isHighValueHidden` when its USD value is greater than `VALUABLE_TOKENS_NUM`. The value is computed as `(price_per_token * uiAmount).toFixed(4)`. When a price is missing, the result becomes the string `"NaN"`, and `Number("NaN")> 10` evaluates to false. The token is therefore not hidden.

These unpriced tokens remain visible and can be selected by Select All. The burn path then collects accounts where `selected && !isHighValueHidden` and executes Burn + Close.

Impact

A user relying on the product's "hide valuable assets" protection may permanently burn assets that did not have reliable pricing at scan time. This is a failure of an irreversible-action safety guard, not a fee issue.

Recommendation

1. Treat missing or non-numeric prices as high risk and exclude those tokens from Select All.
2. Require item-by-item explicit confirmation before burning unpriced tokens.

3. Allow one-click burn selection only when a reliable price exists and is below the configured threshold.

Client Response

Tokens with missing, abnormal, or non-computable USD values are now marked as high risk and are no longer included in bulk Select All selection. Users can still expand the view and manually select such tokens, but they are not selected automatically for irreversible burn actions.

Low Severity

[L-01] Malicious NFT symbols can cause persistent denial of service

Severity	Status
Low	Fixed

Location:

- `src/assets/hooks/useCleanUpFeatures.ts:313, 349-363`

Description

NFT symbol values come from untrusted DAS metadata, but they are used directly as keys in a plain object.

Proof of Concept

```
1 if (groups[symbol] == undefined) groups[symbol] = [];
2 groups[symbol].push(nft);
```

When `symbol` is `__proto__`, `constructor`, or `toString`, the lookup can hit inherited properties. The `== undefined` check evaluates to false, and the subsequent `.push` call is made on a non-array value, consistently throwing a `TypeError`. A local minimal reproduction confirms this behavior for the listed values.

An attacker can airdrop an asset with one of these symbols to a target wallet and make every NFT scan fail. The exception occurs before `setIsCheckingNFTAccounts(false)`, and the current `catch` path does not reset that state.

Impact

The impact is a low-cost, persistent denial of service against the NFT cleanup feature. It does not directly lead to fund theft.

Recommendation

1. Use `Map<string, NFT[]>` or `Object.create(null)` for grouping.
2. Use `Object.hasOwn` for key checks.
3. Isolate malformed assets so one bad asset does not abort the full scan.

Client Response

NFT grouping by symbol now uses a no-prototype object and `Object.hasOwn` for key checks. This prevents malicious `__proto__`, `constructor`, or `toString` symbols from hitting inherited object properties, and the exception path now resets the NFT scanning loading state.

Informational Severity

[I-01] Helius RPC key is exposed in the browser and can be abused for quota theft

Severity	Status
Informational	Fixed

Location:

- `vite.config.mts:29`
- `constants/index.ts:15, 44`

Description

`ENV_SOL_RPC_URL` is injected into the client through the Vite environment prefix and appears in the final JavaScript / RPC URL. Third parties can copy the key and consume paid request quota or rate limits, causing asset scanning and transaction RPC calls to fail. This can also increase billing exposure.

The submitted evidence includes a real API key screenshot.

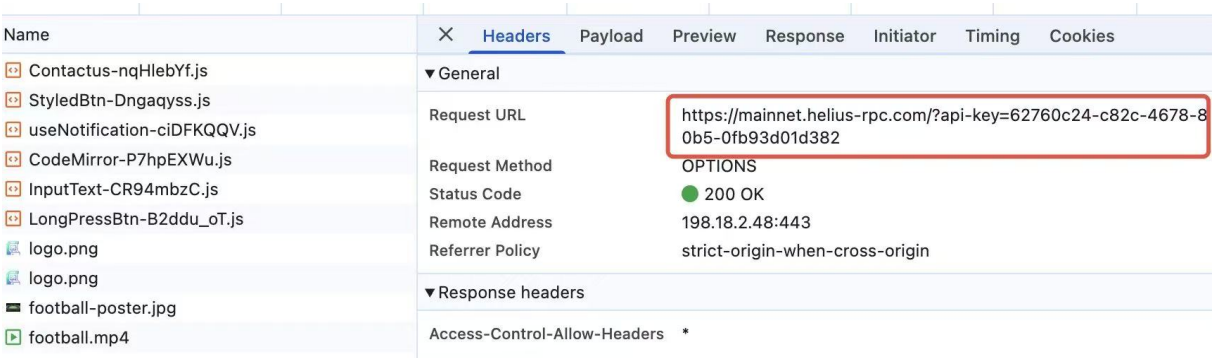

Figure 2: RPC key exposure evidence

Recommendation

Create a browser-specific low-privilege key, enforce domain, method, and rate restrictions, enable alerting, and route critical RPC calls through a controlled proxy.

Client Response

The frontend no longer directly contains the Helius key. RPC requests now go through the games backend `/api/rpc` reverse proxy, while the browser only holds a low-privilege `X-`

`RPC-Proxy-Key` for proxy authentication. The real Helius key is stored only in backend infrastructure.

Risk Classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Severity	Description
Critical	May directly cause loss of funds, permanent system failure, or complete compromise of a critical trust assumption.
High	May cause material asset loss, privilege escalation, or serious system malfunction under realistic conditions.
Medium	May cause limited loss, degraded security guarantees, or incorrect behavior when specific preconditions are met.
Low	Has limited direct impact, exposes a defensive gap, or concerns edge-case correctness or hardening.
Informational	Improves clarity, maintainability, operational safety, or documentation without immediate exploit impact.
Gas	Reduces execution cost without changing intended behavior.

About ZeroDrift

ZeroDrift is the AI security layer for high stakes software. We combine expert security judgment with AI-assisted analysis to identify, explain, and help remediate software risk.

Disclaimer

This report documents a time-boxed, point-in-time assessment of only the systems and versions listed in Audit Scope. It does not guarantee the absence of vulnerabilities and is not an endorsement of the underlying business, product, or deployment. Security depends on implementation, configuration, operations, integrations, and subsequent changes that may fall outside this assessment.