



Security Assessment

WiSP

Verified on October 24, 2026

SUMMARY

Project

WISP

CHAIN

N/A

METHODOLOGY

Manual & Automatic Analysis

FILES

Repository

DELIVERY

October 24, 2026

TYPE

Standard Audit



20

0

1

4

13

2

20

Total Findings

Critical

Major

Medium

Minor

Informational

Resolved

0 Critical

An exposure that can affect the contract functions in several events that can risk and disrupt the contract

1 Major

An opening & exposure to manipulate the contract in an unwanted manner

4 Medium

An opening that could affect the outcome in executing the contract in a specific situation

13 Minor

An opening but doesn't have an impact on the functionality of the contract

2 Informational

An opening that consists information but will not risk or affect the contract

20 Resolved

ContractWolf's findings has been acknowledged & resolved by the project

STATUS

✓ AUDIT PASSED

TABLE OF CONTENTS | WiSP

Summary

Project Summary
Findings Summary
Disclaimer
Scope of Work
Auditing Approach

Project Information

Token/Project Details

Findings

Issues
Penetration Attacks
Fixes & Recommendation
Audit Comments

DISCLAIMER | WiSP

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project dapp" to "External dapp" such as PancakeSwap, UniSwap, SushiSwap or similar.

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **DAPP**.

ContractWolf's presence is to analyze, audit and assess the Client's files to find any underlying risk and to eliminate any logic and flow errors within its code.

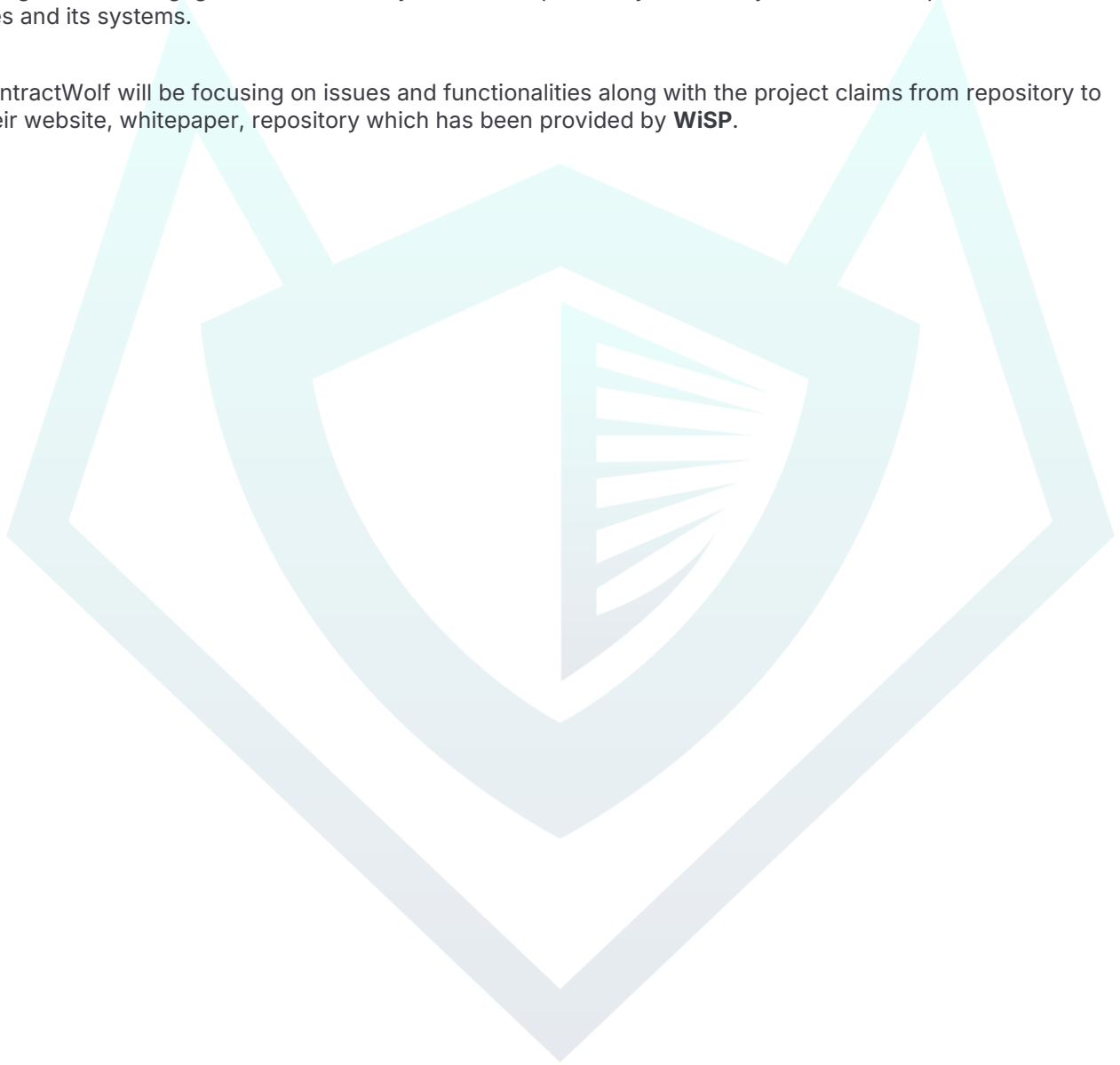
Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | WiSP

WiSP team has agreed and provided us with the files that need to be tested (*Github, BSCscan, Etherscan, Local files etc*). The scope of audit is the main Repository.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of the files and its systems.

ContractWolf will be focusing on issues and functionalities along with the project claims from repository to their website, whitepaper, repository which has been provided by **WiSP**.



AUDITING APPROACH | WiSP

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and contract scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

The auditing process follows a routine series of steps :

1. Code review that includes the following :
 - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the files.
 - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.
2. Testing and automated analysis that includes :
 - Testing the files function with common test cases and scenarios to ensure that it returns the expected results.
3. Best practices and ethical review. The team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the dapp.
4. Recommendations to help the project take steps to eliminate or minimize threats and secure their files.

TOKEN DETAILS | WiSP



WiSP is a privacy layer for on-chain trading, built to make private trading as fast and easy as trading from a normal wallet. Shield, buy, sell, and move funds without your trades being linked back to you and stay in control of your funds throughout. Privacy made easy.

SOURCE

Source

wisp-audit-cw commit id : 7ac8cf07f6b97a58a3665edf10c8d16943b7c1bb

FINDINGS | WiSP



20

0

1

4

13

2

20

Total Findings

Critical

Major

Medium

Minor

Informational

Resolved

This report has been prepared to state the issues and vulnerabilities for WiSP through this audit. The goal of this report findings is to identify specifically and fix any underlying issues and errors

ID	Title	File & Line #	Severity	Status
RAIL-EXIT-LINKABLE	Every cross-chain exit publishes the account's funding address	src/app-main.mjs, L: 1873	Major	● Resolved
DCW-017	Rate limits key on a request header the client controls	server/app-server.mjs, L: 30	Medium	● Resolved
DCW-011	The exit pays a deposit address from an unverified quote	src/rail.mjs, L: 121	Medium	● Resolved
INSTANT-LINKABILITY	The default profile funds a trading wallet seconds after a shield	src/privacy-policy.mjs, L: 43	Medium	● Resolved
DCW-027	WiSP's Helius API key is compiled into the public bundle	vite.config.mjs, L: 62	Medium	● Resolved
JOURNAL-UNBOUNDED	The operation journal is never pruned, so saves eventually fail	src/vault.mjs, L: 23	Minor	● Resolved
DCW-003	HSTS lasts one day and the planned ramp was never completed	scripts/security-headers.mjs, L: 13	Minor	● Resolved
DCW-016	The customer's RPC provider key is stored unencrypted	src/app-main.mjs, L: 1400	Minor	● Resolved
DCW-014	A position's mint, cost and mode are never type-checked	src/model.mjs, L: 30	Minor	● Resolved
DCW-009	The payout spending limit must allow any destination	server/payouts.mjs, L: 451	Minor	● Resolved

DCW-019	A page restored from the back-forward cache returns unlocked	src/app-main.mjs, L: 1055	Minor	● Resolved
DCW-007	25 open dependency advisories, one of them high	package-lock.json, L: 1984	Minor	● Resolved
DCW-024	Mutable base image and an unverified prebuilt bundle	Dockerfile, L: 6	Minor	● Resolved
FEE-BAND-CLIENT	The platform fee band is chosen and enforced only in the client	src/identity.mjs, L: 76	Minor	● Resolved
BACKUP-KDF-WEAK	The exported backup protects the phrase with a weak key derivation	src/vault.mjs, L: 16	Minor	● Resolved
PROVIDER-FEE-UNCAPPED	The pool provider's live config sets the withdrawal fee with no ceiling	src/customer-controller.mjs, L: 1078	Minor	● Resolved
PHRASE-SPELLCHECK	The recovery phrase field allows browser spellcheck and autocorrect	app.html, L: 69	Minor	● Resolved
LAB-ON-PROD	The developer lab and the acceptance host are still public	vite.config.mjs, L: 81	Minor	● Resolved
DCW-018	Circulating supply is converted through Number without a bound	server/supply.mjs, L: 24	Informational	● Resolved
CSP-RPC-WILDCARD	The CSP's RPC wildcards give a compromised script somewhere to send	scripts/security-headers.mjs, L: 5	Informational	● Resolved

PENETRATION ATTACKS | WiSP

Dapp Weakness Classification and Test Cases

ID	Name	Description	Status
DCW-001	Malware Scan	Checks whether the dApp domain, hosted files, or third-party scripts distribute malicious code capable of compromising user devices or connected wallets.	● Passed
DCW-002	Phishing	Verifies that the dApp does not impersonate a legitimate service or host content designed to trick users into revealing seed phrases, private keys, or credentials.	● Passed
DCW-003	Missing HTTP Headers	Identifies absent security headers such as CSP, HSTS, X-Frame-Options, and X-Content-Type-Options that leave the application exposed to clickjacking, MIME sniffing, and injection attacks.	● Passed
DCW-004	Valid SSL Certificate	Confirms the domain is served over a properly issued, unexpired, and correctly configured TLS certificate so that user traffic cannot be intercepted or tampered with.	● Passed
DCW-005	Firewalls(Drop and Deny)	Assesses whether firewall or WAF rules are configured to drop and deny malicious traffic, unauthorized ports, and automated attack attempts.	● Passed
DCW-006	Potential SQL Injection	Detects input fields or API parameters that pass unsanitized data into backend database queries, allowing an attacker to read, modify, or destroy stored data.	● Passed
DCW-007	Framework Version	Checks whether the frameworks, libraries, and dependencies in use are outdated or running versions with publicly disclosed vulnerabilities.	● Passed
DCW-008	Gas Griefing	Reviews whether transactions can be manipulated to forward insufficient gas to sub-calls, causing operations to fail silently while the parent transaction still reports success.	● Passed
DCW-009	Address Approval	Examines token approval requests to ensure the dApp does not request unlimited or unnecessary allowances that could later be abused to move user funds.	● Passed
DCW-010	Address Draining	Identifies logic or contract calls that could transfer tokens, NFTs, or native assets out of a connected wallet without the informed consent of the owner.	● Passed
DCW-011	Insecure API Usage	Evaluates whether APIs are consumed over insecure channels, without proper authentication, or with keys and endpoints exposed on the client side.	● Passed
DCW-012	Error Handling	Verifies that application errors and failed transactions are caught and handled gracefully without exposing stack traces, internal paths, or leaving the interface in an unstable state.	● Passed
DCW-013	Memory Leak	Detects client-side code that fails to release memory from listeners, intervals, or subscriptions, degrading performance or crashing the browser during prolonged sessions.	● Passed
DCW-014	Lack of Input Validation	Checks whether user-supplied values are validated for type, format, range, and length on both the client and server before being processed or submitted on-chain.	● Passed
DCW-015	Potential Backdoor	Looks for hidden functions, privileged accounts, or undocumented endpoints that allow the owner or a third party to bypass normal access controls.	● Passed
DCW-016	Sensitive Data Exposure	Identifies secrets, private keys, API credentials, or personal information exposed through source code, network responses, logs, or browser	● Passed

ID	Name	Description	Status
		storage.	
DCW-017	Request Limit	Assesses whether rate limiting and throttling are enforced to prevent brute force, spam, and denial-of-service abuse against application endpoints.	● Passed
DCW-018	Overflow or Precision Loss	Reviews arithmetic operations and token decimal handling for overflow, underflow, or rounding errors that could misstate balances and transaction amounts.	● Passed
DCW-019	Unintended Behavior	Covers functionality that deviates from the documented or expected flow, producing unexpected results, inconsistent state, or avoidable risk for the user.	● Passed
DCW-020	Cross-Site Scripting (XSS)	Checks whether user-controlled input is rendered into the page without escaping, allowing injected scripts to run in a visitor's session and reach their wallet.	● Passed
DCW-021	Cross-Site Request Forgery	Checks whether state-changing requests can be triggered from another origin without anti-forgery tokens or SameSite cookie protection.	● Passed
DCW-022	Wallet Signature Phishing	Reviews the messages and transactions the dApp asks users to sign, flagging blind signing, unlimited permits and requests that do not match the action shown.	● Passed
DCW-023	Content Security Policy and Subresource Integrity	Checks that scripts and styles load only from approved origins and that third-party files are pinned with integrity hashes.	● Passed
DCW-024	Supply Chain and Dependency Risk	Scans front-end packages and CDN-hosted libraries for known vulnerabilities, abandoned maintainers and tampered builds.	● Passed
DCW-025	Domain and DNS Hygiene	Reviews DNS records, registrar locks, DNSSEC and dangling subdomains that could be hijacked to serve a malicious copy of the dApp.	● Passed
DCW-026	RPC Endpoint Trust	Checks which RPC and node providers the dApp trusts, and whether a malicious or compromised endpoint could return falsified balances or state.	● Passed
DCW-027	Exposed Secrets and Keys	Searches shipped bundles, source maps and repositories for API keys, private keys and credentials left in client-side code.	● Passed
DCW-028	Broken Access Control	Verifies that admin routes, internal APIs and user data cannot be reached or modified by changing identifiers or skipping the interface.	● Passed
DCW-029	Clickjacking	Checks that the dApp cannot be framed by another site to trick users into approving transactions they cannot see.	● Passed
DCW-030	Transaction Simulation Mismatch	Compares what the interface displays with what the wallet will actually execute, covering amounts, recipients, slippage and spender addresses.	● Passed

FIXES & RECOMMENDATION

RAIL-EXIT-LINKABLE | Every cross-chain exit publishes the account's funding address

A cross-chain exit asks NEAR Intents for a quote with `refundTo` set to the account's funding address, the public address that made its shields. The pool withdrawal then pays the quote's `depositAddress`, which is visible on chain. 1Click's `/v0/status` answers anyone without a key and returns the original `quoteRequest`, so any observer reads the funding address and the destination beside the withdrawal the card calls private.

```
// src/app-main.mjs: every exit names the funding address as its refund target
const body = quoteBody({ dir: 'out', asset: a, amount: net, recipient,
  refundTo: lastView.fundingAddress, dry: false });

// GET /v0/status?depositAddress=<pool withdrawal recipient>, no key required:
// { quoteResponse: { quoteRequest: { refundTo, recipient, ... }, quote: { ... } } }
```

Recommendation

Refund each exit to a fresh wallet derived for that exit, never the funding address, and request the quote with `confidentiality` set, so neither the service nor its public status endpoint can join the withdrawal to the account.

```
// A one-time refund wallet per exit: a refund lands unlinked and is shielded again.
const refund = await call('newRefundWallet', { railId });
const body = quoteBody({ dir: 'out', asset: a, amount: net, recipient,
  refundTo: refund.address, dry: false });
body.confidentiality = 'basic'; // deposit and payout cannot be matched
```

DCW-017 | Rate limits key on a request header the client controls

`clientAddress()` reads `fly-client-ip` from the request headers and only falls back to the socket peer, with no check that a trusted proxy set it. Every limiter keys on that value, so a caller that sends a fresh header per request takes an unbounded share of the referral write API, the daily invite allowance and the price route. The container listens on `0.0.0.0`, so any peer reaching the port directly bypasses the edge.

```
const clientAddress = req =>
  String(req.headers['fly-client-ip'] ?? req.socket.remoteAddress ?? '');
```

Recommendation

Trust the header only when the socket peer is the platform proxy, and fall back to `req.socket.remoteAddress` otherwise, so a forged header shares the real peer's bucket.

```
const PROXY_CIDR = process.env.WISP_TRUSTED_PROXY_CIDR ?? '';
const viaProxy = req =>
  PROXY_CIDR !== '' && inCidr(req.socket.remoteAddress, PROXY_CIDR);

const clientAddress = req => {
  const peer = String(req.socket.remoteAddress ?? '');
  const claimed = String(req.headers['fly-client-ip'] ?? '');
  return viaProxy(req) && claimed ? claimed : peer;
};
```

DCW-011 | The exit pays a deposit address from an unverified quote

`parseQuote()` checks that the echoed request matches and that `depositAddress` has the shape of a Solana address, then the pool withdrawal pays it. The comment says the quote's signature cannot be checked, but 1Click documents `verifyQuoteSignature()` for exactly this field. Any party able to alter the response, such as an intercepting proxy or a compromised script, can substitute its own address, and the person approving cannot tell the difference.

```
// the service is trusted for the deposit address itself (its quote carries a  
// signature this page has no key to check)  
if (!(toUser ? validSolanaAddress(q.depositAddress)  
      : validDepositAddress(asset, q.depositAddress)))  
  throw new Error('Cross-chain deposit address invalid');
```

Recommendation

Verify the quote's signature over `quoteRequest`, `quote` and `timestamp` with 1Click's published key before showing or paying `depositAddress`, and verify status payloads the same way.

```
import { verifyQuoteSignature } from '@defuse-protocol/one-click-sdk-typescript';  
  
const json = await railCall(fetch, '/v0/quote', { method: 'POST', body });  
if (!verifyQuoteSignature(json)) throw new Error('Quote not signed by 1Click');  
const q = parseQuote(json, { dir: 'out', asset: a, body });
```

INSTANT-LINKABILITY | The default profile funds a trading wallet seconds after a shield

New accounts start on the instant profile, whose funding floor and jitter are both zero, so `fundingOpensAt()` opens a withdrawal at the moment of the shield that credited the pool, with only the two slots of `MINIMUM_SLOT_SPACING` between them. At the roughly 42 pool transactions an hour the project measured, a withdrawal that close to a deposit has almost no crowd, which links the funding address to the trading wallet and its trades.

```
export const MINIMUM_SLOT_SPACING = 2;
export const PRIVACY_PROFILES = Object.freeze({
  instant: Object.freeze({ fundingFloorMs: 0, fundingJitterMs: 0 }),
  ...
});
export const DEFAULT_PRIVACY = Object.freeze({ profile: 'instant', customMinutes: 10 });
```

Recommendation

Default new accounts to steady, and keep a floor between a pool credit and the first withdrawal it funds on every profile, with instant reached only through an explicit choice that names the timing link it creates.

```
export const PRIVACY_PROFILES = Object.freeze({
  instant: Object.freeze({ fundingFloorMs: 2 * 60_000, fundingJitterMs: 3 * 60_000 }),
  steady: Object.freeze({ fundingFloorMs: 5 * 60_000, fundingJitterMs: 10 * 60_000 }),
  max: Object.freeze({ fundingFloorMs: 30 * 60_000, fundingJitterMs: 30 * 60_000 }),
});
// New accounts start private; Instant is an explicit, disclosed choice.
export const DEFAULT_PRIVACY = Object.freeze({ profile: 'steady', customMinutes: 10 });
```

DCW-027 | WiSP's Helius API key is compiled into the public bundle

`defaultRpcFor()` builds `https://mainnet.helius-rpc.com/?api-key=<key>` into `__WISP_DEFAULT_RPC__`, so the key ships in the worker bundle every visitor downloads. Its only fence is a domain restriction, which Helius checks against the `Origin` header, and any non-browser client sets that header freely: `server/watch.mjs` does exactly that. Anyone can drain the credit budget, and every customer on the default RPC then fails.

```
// vite.config.mjs: the key becomes a string constant in the public bundle
return `https://mainnet.helius-rpc.com/?api-key=${key}`;

// server/watch.mjs: a server passes the domain restriction by naming the origin
const r = await fetch(RPC, { method: 'POST',
  headers: { 'content-type': 'application/json', origin: ORIGIN } });
```

Recommendation

Hold the key server side behind a same-origin proxy that rate limits and allows only the methods the app uses, or drop the shared key and ask each customer for an endpoint, since the domain restriction is not an access control.

```
// server: forward only the methods the app needs; the key never leaves the server
const ALLOWED = new Set(['getAccountInfo', 'getMultipleAccounts', 'getLatestBlockhash',
  'simulateTransaction', 'getSignatureStatuses', 'sendTransaction']);
if (!ALLOWED.has(body.method)) return json(res, 400, { error: 'Method not allowed' });
const upstream = await fetch(`https://mainnet.helius-rpc.com/?api-key=${HELIUS_KEY}`,
  { method: 'POST', headers: { 'content-type': 'application/json' }, body: raw });
```

JOURNAL-UNBOUNDED | The operation journal is never pruned, so saves eventually fail

state.operations and state.relayedWithdrawals gain a record per transaction and are never pruned: there is no cap and no removal of terminal records. A wallet-signed deposit keeps its intent.unsignedBase64, so records run to kilobytes. Once the sealed state passes MAX_PLAINTEXT_BYTES, seal() throws and no further settlement can be written, and the message tells the user to archive records that no code can archive.

```
// src/vault.mjs: the only bound, and nothing in the app can relieve it
if (plaintext.length > MAX_PLAINTEXT_BYTES)
  throw new Error('Vault too large to reopen; archive old journal records');

// src/customer-controller.mjs: every save re-seals the whole journal
const save = async next => { validateState(next); await write(await seal(next, pw)); };
```

Recommendation

Drop terminal operations past a retention bound as part of saving, keeping every unresolved record and a short tail for history, so the sealed state stays well under MAX_PLAINTEXT_BYTES.

```
const JOURNAL_KEEP = 200; // terminal records retained for history

function pruneJournal(state) {
  const live = state.operations.filter(o => !CLIENT_TERMINAL.includes(o.status));
  const done = state.operations.filter(o => CLIENT_TERMINAL.includes(o.status));
  state.operations = [...done.slice(-JOURNAL_KEEP), ...live];
  return state;
}

const save = async next => { await write(await seal(pruneJournal(next), pw)); };
```

DCW-003 | HSTS lasts one day and the planned ramp was never completed

productionHeaders sends Strict-Transport-Security: max-age=86400, one day, with no preload. The comment beside it plans one day from 2026-10-15 while the domain settles and then a year, but the value was never raised. A browser whose pin has lapsed, or that has never visited, can be moved to plain http on first contact, which is the load that carries the recovery phrase screen.

```
'Strict-Transport-Security': 'max-age=86400; includeSubDomains',
```

Recommendation

Raise max-age to 31536000 with preload once nothing under the domain needs plain http, then submit it to the preload list. Add Cross-Origin-Opener-Policy while editing this list.

```
'Strict-Transport-Security': 'max-age=31536000; includeSubDomains; preload',  
'Cross-Origin-Opener-Policy': 'same-origin',  
'Cross-Origin-Resource-Policy': 'same-origin',
```

DCW-016 | The customer's RPC provider key is stored unencrypted

A custom RPC endpoint is written to `localStorage` verbatim, outside the sealed vault. Provider URLs carry the credential in the path or the query, which `maskEndpoint()` hides on screen precisely because it is a secret. Anything able to read the origin's storage, including a shared or recovered browser profile, reads the key and spends the customer's quota.

```
// the same value the interface deliberately masks for display
try { localStorage.setItem(RPC_STORAGE, view.rpcEndpoint); } catch { /* ignore */ }

const path = u.pathname.split('/')
  .map(seg => /^[A-Za-z0-9_-]{16,$}/.test(seg) ? 'x' : seg).join('/');
```

Recommendation

Keep the endpoint inside the encrypted vault with the rest of the account, or store only its origin and ask for the credential once per session.

```
// The endpoint joins the sealed account state; localStorage holds no secret.
await call('setRpc', { url: endpoint, remember: true });

// worker side, inside the vault:
next.customer.rpcEndpoint = endpoint;
await save(next);
```

DCW-014 | A position's mint, cost and mode are never type-checked

`validateState()` checks each position's status, index and balance, and the elements of holdings, leftTokens and previousMints, but never `p.mint`, `p.cost` or `p.mode`. A backup that opens under the owner's own password is therefore adopted with a non-string mint, and the screens that compare and render it then fail.

`simulate()` reads `BigInt(p.cost)` and `MODES[p.mode]` with no guard.

```
for (const p of state.positions) {
  if (!['funded', 'holding', 'pending', 'retired'].includes(p.status)
    || !Number.isSafeInteger(p.index) || p.index < 0 || p.index >= state.nextIndex
    || !/^\d+$/.test(p.balance)) throw new Error('Invalid position');
  // p.mint, p.cost and p.mode are never checked
}
```

Recommendation

Type-check the scalar fields beside the collections already validated: mint a string or null, cost an optional digit string and mode a known key, so a malformed record is refused at the boundary.

```
for (const p of state.positions) {
  if (!['funded', 'holding', 'pending', 'retired'].includes(p.status)
    || !Number.isSafeInteger(p.index) || p.index < 0 || p.index >= state.nextIndex
    || !/^\d+$/.test(p.balance)) throw new Error('Invalid position');
  if (p.mint !== undefined && p.mint !== null && typeof p.mint !== 'string')
    throw new Error('Invalid position');
  if (p.cost !== undefined && !/^\d+$/.test(String(p.cost)))
    throw new Error('Invalid position');
  if (p.mode !== undefined && !Object.hasOwn(MODES, p.mode))
    throw new Error('Invalid position');
}
```

DCW-009 | The payout spending limit must allow any destination

Startup refuses the payout limit unless `limit.destinations.length` is zero, so the Squads limit is required to permit any destination. The worker key is its only member and the limit is a standing delegation, so a stolen key moves 1 SOL a day out of the referral vault to an address of the attacker's choosing, with no second signature from the 2 of 3 multisig.

```
if (limit.multisig !== c.multisig || !limit.sol || limit.members.length !== 1
    || limit.members[0] !== signer.address || limit.destinations.length)
  throw new Error('The spending limit does not match this worker');
```

Recommendation

Pay through a destination-locked hop, or rotate a short destination list onto the limit as each batch is planned, so a stolen key reaches only addresses the owner has already approved.

```
// Every payee of this run must already be named on the limit.
const approved = new Set(limit.destinations);
if (!approved.size) throw new Error('The payout limit must name its destinations');
for (const p of payments)
  if (!approved.has(p.payee)) throw new Error('Payee is not an approved destination');
```

DCW-019 | A page restored from the back-forward cache returns unlocked

Lock is `location.reload()`, which clears the unlocked session, but the app registers no `pageshow`, `pagehide` or `visibilitychange` handler. Navigating away and pressing Back restores the document from the back-forward cache with `unlocked` still true and the decrypted account live in the worker, so the next person at the device sees the open account without the password.

```
// the only lock path; no pageshow or pagehide handler exists
$('lock').onclick = () => { location.reload(); };
```

Recommendation

Add a `pageshow` handler that reloads when `event.persisted` is true, and lock on `pagehide`, so a restored document always comes back at the password screen.

```
$('lock').onclick = () => { location.reload(); };

// A restored document is not a session the user is still attending.
addEventListener('pageshow', e => { if (e.persisted) location.reload(); });
addEventListener('pagehide', () => { if (unlocked) worker?.postMessage({ lock: 1 }); });
```

DCW-007 | 25 open dependency advisories, one of them high

npm audit reports 25 advisories against the locked dependency tree, among them a high in bigint-buffer 1.1.5, a buffer overflow in its native toBigIntLE, and that package carries an install script. The project's own analysis finds none reachable on a money path, but that analysis is the only control: no build or release step fails when a new or newly reachable advisory appears.

```
"node_modules/bigint-buffer": {
  "version": "1.1.5",
  "resolved": "https://registry.npmjs.org/bigint-buffer/-/bigint-buffer-1.1.5.tgz",
  "hasInstallScript": true,
```

Recommendation

Replace bigint-buffer with a pure JavaScript implementation where the stack allows, and fail the release build on any high or critical advisory that has no recorded reachability review, so a new one stops the release instead of waiting for a manual audit.

```
// scripts/check-gate.mjs, before a release is accepted
const ACCEPTED = 'audit/accepted-advisories.json'; // each with its reachability note
const run = spawnSync('npm', ['audit', '--json', '--omit=dev'], { encoding: 'utf8' });
const reviewed = new Set(JSON.parse(readFileSync(ACCEPTED, 'utf8')));
for (const [name, a] of Object.entries(JSON.parse(run.stdout).vulnerabilities ?? {}))
  if (['high', 'critical'].includes(a.severity) && !reviewed.has(name))
    throw new Error(`Unreviewed ${a.severity} advisory in ${name}; release refused`);
```

DCW-024 | Mutable base image and an unverified prebuilt bundle

The image builds FROM `node:24-slim`, a mutable tag that resolves to whatever was last pushed, and COPY `dist ./dist` takes a prebuilt bundle checked only for the presence of four files. Nothing in the build ties the served bundle, which holds the signing code customers run, to a reviewed commit, so a stale or substituted `dist/` ships without failing anything.

```
FROM node:24-slim
COPY dist ./dist
RUN test -f dist/app.html && test -f dist/tokens.json && test -d dist/icons
```

Recommendation

Pin the base image by digest, and either build `dist/` inside the image from the reviewed commit or verify its release manifest hash in the RUN step, so only the reviewed bundle can be served.

```
FROM node:24-slim@sha256:<pinned digest>
COPY dist ./dist
COPY release-manifest.sha256 ./
# The bundle must be the reviewed one, not merely present.
RUN sha256sum --check release-manifest.sha256
```

FEE-BAND-CLIENT | The platform fee band is chosen and enforced only in the client

`validFeeBps()` only tests membership in a closed set, and the fee itself is a `SystemProgram.transfer` the client builds into its own trade transaction. No program, server or signature binds the chosen band to the holding or invite that earns it, so a modified client pays the lowest band, or omits the transfer, and the trade still settles.

```
export function validFeeBps(bps) {
  if (!ALLOWED_FEE_BPS.has(bps)
    && !(__WISP_FEATURE_TIERS__ && TIER_FEE_BPS.includes(bps)))
    throw new Error('Invalid fee band');
  return bps; // membership only: nothing proves the band was earned
}
```

Recommendation

Have the treasury co-sign the band it will accept for a wallet and window, or reconcile the fee transfers against the tier reads afterwards, so an unearned band is detected rather than assumed absent.

```
// The treasury signs the band it will accept for this wallet and window.
export function assertFeeBandProof(bps, proof, trustedKey) {
  const message = `WiSP fee band ${bps} for ${proof.wallet} to ${proof.expiry}`;
  if (proof.expiry < Date.now()) throw new Error('Fee band proof expired');
  if (!verifySignature(message, trustedKey, proof.signature))
    throw new Error('Unearned fee band');
  return validFeeBps(bps);
}
```

BACKUP-KDF-WEAK | The exported backup protects the phrase with a weak key derivation

The backup file is `seal(controller.snapshot(), password)`, the whole account including phrase, and the export hands it to the share sheet or a download, so it is meant to leave the device. Its only defense against offline guessing is PBKDF2-SHA256 at 310000 rounds, about half the current OWASP figure, under a password checked for length alone, so `aaaaaaaaaaaa` is accepted and a dictionary password falls quickly on a GPU.

```
if (typeof password !== 'string' || password.length < 12)
  throw new Error('Use a password of at least 12 characters');
return crypto.subtle.deriveKey(
  { name: 'PBKDF2', hash: 'SHA-256', salt, iterations: 310000 }, material,
  { name: 'AES-GCM', length: 256 }, false, ['encrypt', 'decrypt']);
```

Recommendation

Raise PBKDF2 to at least 600000 rounds, or move the export to Argon2id, and refuse passwords that a strength estimate such as `zxcvbn` scores as guessable, keeping the version field so older backups still open.

```
export const KDF_V2 = { name: 'PBKDF2', hash: 'SHA-256', iterations: 600000 };

async function key(password, salt, kdf = KDF_V2) {
  if (typeof password !== 'string' || password.length < 12 || zxcvbn(password).score < 3)
    throw new Error('Choose a longer, less guessable password');
  const material = await crypto.subtle.importKey('raw', enc.encode(password), 'PBKDF2',
    false, ['deriveKey']);
  return crypto.subtle.deriveKey({ ...kdf, salt }, material,
    { name: 'AES-GCM', length: 256 }, false, ['encrypt', 'decrypt']);
}
```

PROVIDER-FEE-UNCAPPED | The pool provider's live config sets the withdrawal fee with no ceiling

Funding and exits compute the fee as `withdrawalFee(gross, config)` from the provider's live `/config` response. `withdrawalFee()` refuses only a rate of 100% or more and a fixed part above 1 SOL, and the relayer journal enforces a minimum fee, never a maximum. A provider that raises its rate, or a tampered response, takes the difference out of the customer's withdrawal, checked only by the person reading the approval sheet.

```
// src/customer-controller.mjs: the fee is whatever the provider's config says
const config = await providerJson('config'), fee = Number(withdrawalFee(gross, config));

// src/quotes.mjs: the only bound on that schedule
if (rate >= SOL || rent > SOL) throw new Error('Unsupported provider fee schedule');
```

Recommendation

Pin the reviewed schedule, 0.35% plus 0.006 SOL, as a ceiling beside the minimum the journal already enforces, and refuse or re-confirm any quote above it, so a fee change is a reviewed release rather than a remote setting.

```
// The schedule WiSP reviewed: a change upstream is a release, not a remote setting.
const MAX_RATE = 3_500_000n, MAX_RENT = 6_000_000n; // 0.35% in 1e9 units, 0.006 SOL

export function withdrawalFee(grossLamports, config) {
  const rate = decimal(config?.withdraw_fee_rate, 9);
  const rent = decimal(config?.withdraw_rent_fee, 9);
  if (rate > MAX_RATE || rent > MAX_RENT)
    throw new Error('Provider fee is above the reviewed schedule');
  return BigInt(grossLamports) * rate / SOL + rent;
}
```

PHRASE-SPELLCHECK | The recovery phrase field allows browser spellcheck and autocorrect

The textarea where a customer types or pastes the 24 words sets `autocomplete="off"` but not `spellcheck="false"`, `autocorrect="off"` or `autocapitalize="none"`, which the address fields on the same page do set. With Chrome's enhanced spell check or Edge's Microsoft Editor turned on, the field's text is sent to the vendor's servers, and mobile keyboards may learn the words into their suggestions.

```
<label>Recovery phrase<textarea id="phrase" rows="2" autocomplete="off"
placeholder="24 words, WiSP phrase only ..."></textarea></label>
```

Recommendation

Mark the phrase field, and the lab's, with `spellcheck="false"`, `autocorrect="off"`, `autocapitalize="none"` and `autocomplete="off"`, as the address fields already are, so its contents never reach a cloud spellchecker or a keyboard dictionary.

```
<label>Recovery phrase<textarea id="phrase" rows="2" autocomplete="off"
spellcheck="false" autocorrect="off" autocapitalize="none"
placeholder="24 words, WiSP phrase only ..."></textarea></label>
```

LAB-ON-PROD | The developer lab and the acceptance host are still public

The build includes lab: 'index.html', and the server serves any root page, so app.usewisp.trade/index.html answers HTTP 200 with the integration lab: its own phrase form, backup import and wallet connect, running on the same origin as the customer vault. test.usewisp.trade also still answers, an acceptance host with its own referral API that fly.acceptance.toml says to destroy after acceptance.

```
// vite.config.mjs: the lab is a production build input
rollupOptions: { input: { lab: 'index.html', app: 'app.html',
  landing: 'landing/index.html', security: 'landing/security.html' } }

// fly.acceptance.toml, still serving at test.usewisp.trade
# Destroy after acceptance: flyctl apps destroy <app>.
```

Recommendation

Build the lab only in a development mode and refuse /index.html in the production server, and destroy the acceptance app now that the release is live, so production carries no test surface beside the customer's vault.

```
// vite.config.mjs: the lab is built only outside production
const input = { app: 'app.html', landing: 'landing/index.html' };
if (mode === 'development') input.lab = 'index.html';

// once, from the owner's shell:
// flyctl apps destroy wisp-accept-f4d424
```

DCW-018 | Circulating supply is converted through Number without a bound

`toNumber()` divides `Number(amount)` by `10 ** decimals`, where `units()` accepts up to 30 digits and up to 18 decimals straight from the RPC. Nothing asserts the base-unit count is below `Number.MAX_SAFE_INTEGER`, so only a comment keeps the result exact. A mint with more supply or more decimals than assumed would silently round the figure served to listing services.

```
// "at most 1e9 tokens x 1e6 units: inside Number's exact range"  
const toNumber = (amount, decimals) => Number(amount) / 10 ** decimals;
```

Recommendation

Assert the base-unit total is a safe integer, or format the quotient from the bigint directly, so the published figure is exact for any mint rather than only the expected one.

```
function toNumber(amount, decimals) {  
  const scale = 10n ** BigInt(decimals);  
  const whole = amount / scale, fraction = amount % scale;  
  if (whole > BigInt(Number.MAX_SAFE_INTEGER)) throw unavailable();  
  return Number(`${whole}.${fraction.toString().padStart(decimals, '0')}`);  
}
```

CSP-RPC-WILDCARD | The CSP's RPC wildcards give a compromised script somewhere to send

connect-src admits *.g.alchemy.com, *.quiknode.pro, *.drpc.org and six more multi-tenant RPC hosts, under a comment saying the list is kept short so a compromised dependency has few places to send anything. Anyone can open a free account at these providers and get an endpoint inside the wildcard, and Alchemy's request logs show each request's payload, so a hostile dependency can post the decrypted phrase there and read it back.

```
// "deliberately short so a compromised dependency has few places to send anything"
export const RPC_HOST_ALLOWLIST = Object.freeze(['api.mainnet-beta.solana.com',
  '*.helius-rpc.com', '*.quiknode.pro', '*.rpcpool.com', '*.g.alchemy.com',
  'rpc.ankr.com', '*.drpc.org', '*.shyft.to', '*.syndica.io', '*.chainstack.com']));
```

Recommendation

Narrow the policy to the exact hosts in use and stop treating it as exfiltration containment while any admitted host serves arbitrary accounts; protect the phrase directly by wiping it from worker memory once the keys are derived.

```
// Exact hosts only; a vendor-wide wildcard admits every customer of that vendor.
export const RPC_HOST_ALLOWLIST = Object.freeze(['api.mainnet-beta.solana.com',
  'mainnet.helius-rpc.com']);
// The policy fences hosts, not accounts on a shared host: once the keys are derived,
// keep only the sealed phrase in the worker and drop the plaintext copy.
```

AUDIT COMMENTS | WiSP

Dapp audit comment for a non-technical perspective

- WiSP does not record IP addresses from its users
- Recovery phrase and every private key stay in browser; no WiSP server receives them
- The operator cannot mint, pause, freeze or seize anything in this application.





CONTRACTWOLF

Blockchain Security - Smart Contract Audits