GA GUARDIAN

Aria

IPRWA

Security Assessment

July 30th, 2025

Summary

Audit Firm Guardian

Prepared By Curiousapple, Wafflemakr, Cosine, Osman Özdemir, Michael Lett

Client Firm Aria

Final Report Date July 30, 2025

Audit Summary

Aria engaged Guardian to review the security of their Aria's IPRWA contracts. From the 2nd of July to the 9th of July, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of High and Critical issues detected, Guardian assigns a Confidence Ranking of 2 and recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment.
For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

🔗 Blockchain network: **Story**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianOrg/main-contracts-aria-team1, https://github.com/GuardianOrg/main-contracts-aria-team2, https://github.com/GuardianOrg/main-contracts-aria-fuzz

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Aria
Language	Solidity
Codebase	https://github.com/AriaProtocol/main-contracts
Commit(s)	Initial commit: c12310eb1a7a70ed1a9d4e48bb6c1cd081003dde Final commit: 8a46951b91a8504ea10c87e8e47ec15a61c2141c

Audit Summary

Delivery Date	July 30, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	4	0	0	0	1	3
Medium	6	0	0	0	1	5
Low	14	0	0	2	0	12
Info	3	0	0	0	0	3

Audit Scope & Methodology

Scope and details:
source count: {
total: 2011,
source: 1039,
comment: 607,
single: 327,
block: 280,
mixed: 0,
empty: 365,
todo: 0,
blockEmpty: 0,
commentToSourceRatio: 0.584215591915303}

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Aria, fuzz-testing was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
ARIA-01	Redeemable IPRWA should be equal to the IPRWA balance of the staking contract	✅	❌	✅	10M+
VAULT-01	Whitelist proofs validate correctly	✅	✅	✅	10M+
VAULT-02	Fundraise fractional token balance should increase on successful claim	✅	✅	✅	10M+
VAULT-03	Whitelist fractional token balance increase should match claim amount	✅	✅	✅	10M+
STAKE-01	stIPRWA:IPRWA ratio calculations are consistent	✅	❌	✅	10M+
STAKE-02	Total staked amount = usersNetIPRWADeposited	✅	❌	N/A	10M+
STAKE-03	Stake/Unstake operations preserve token conservation	✅	✅	✅	10M+
STAKE-04	Stake/Unstake operation should not succeed despite actor being blacklisted	✅	✅	✅	10M+
STAKE-05	Staking ratios are inverse of each other	✅	❌	✅	10M+
STAKE-06	User stIPRWA balance changes match stake/unstake amounts	✅	❌	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
STAKE-07	The value that the iprwaPerStIPRWA() function returns should strictly increase never decrease	✅	❌	✅	10M+
STAKE-08	The total amount of IPRWA tokens deposited should always equal IPRWAStaking.usersNetIPRWADeposited	✅	❌	✅	10M+
STAKE-09	Revert reason should never be a under or overflow”	✅	✅	✅	10M+
STAKE-10	Users should always receive > 0 staking tokens after calling stake	✅	❌	✅	10M+
STAKE-11	Users should always receive > 0 IPRWA tokens after calling unstake	✅	❌	✅	10M+
TRANSFER-01	Transfer should not succeed if sender is blacklisted	✅	❌	N/A	10M+
GUIDED-01	Stake-Unstake round-trips should be neutral unless royalties are deposited	✅	❌	✅	10M+
GUIDED-02	Users receive royalties proportional to their stake percentage	✅	✅	✅	10M+
GUIDED-03	Late joiners cannot claim royalties for periods they weren't staking	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Blacklist And License Storage Collisions	Logical Error	High	Resolved
H-02	Inflation Attack In IPRWAStaking	Frontrunning	High	Partially Resolved
H-03	Blacklist Bypass Via Transfer	Validation	High	Resolved
H-04	Stake() Susceptible To Griefing	DoS	High	Resolved
M-01	Incorrect Fractional Token Supply Check	Validation	Medium	Resolved
M-02	Faulty Pause Role Logic	Access Control	Medium	Resolved
M-03	Ineffective License Enforcement	Validation	Medium	Resolved
M-04	Permissionless Vault Creation	Validation	Medium	Resolved
M-05	Incorrect Tracking Of usersNetIPRWADeposited	DoS	Medium	Resolved
M-06	Sandwich Attack Royalties	Gaming	Medium	Partially Resolved
L-01	Signature-Based Mapping Vulnerability	Validation	Low	Resolved
L-02	Smart Wallets Cannot Stake	DoS	Low	Resolved
L-03	Lack Of Slippage Controls	Validation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-04	Front-Running And Value Leakage	Frontrunning	Low	Acknowledged
L-05	Arbitrage Risk From Royalty Deposits	Frontrunning	Low	Acknowledged
L-06	Claims DoS'ed By Admin Fractional Minting	Validation	Low	Resolved
L-07	Warning Regarding Story Registration Fee	Warning	Low	Resolved
L-08	Missing IPAccount execute Functionality	Informational	Low	Resolved
L-09	Fundraise Closed With No Expiration Time Check	Validation	Low	Resolved
L-10	Failed Withdrawals Due To Previous Withdraw	Informational	Low	Resolved
L-11	Merkle Root Updates After Claim	Logical Error	Low	Resolved
L-12	Whitelist Claims DoS'ed	Configuration	Low	Resolved
L-13	recoverLostTokens Blocked By Stale totalDeposits	Logical Error	Low	Resolved
L-14	Blacklisted Tokens Permanently Locked	Warning	Low	Resolved
I-01	Unused Code	Superfluous Code	Info	Resolved
I-02	Incorrect Natspec	Documentation	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-03	Whitelist Vault Claims Can Fail	Configuration	Info	Resolved

H-01 | Blacklist And License Storage Collisions

Category	Severity	Location	Status
Logical Error	High	BlacklistStorage.sol: 12	Resolved

Description PoC

The `Legal` contract inherits from both `Blacklist` and `License` contracts. However, both inherited contracts use the slot 0 to store data, causing data corruption due to the overwritten storage slot.

For example, adding a user to the blacklist will make `licenseVersion` value equal to the `blacklist` length, and vice versa.

Recommendation

Make sure the correct storage identifier is used instead of `0x0`:

• LicenseStorage: `keccak256(abi.encode(uint256(keccak256("aria-protocol.License")) - 1)) & ~bytes32(uint256(0xff));`

• BlacklistStorage: `keccak256(abi.encode(uint256(keccak256("aria-protocol.Blacklist")) - 1)) & ~bytes32(uint256(0xff)):`

Resolution

Aria Team: The issue was resolved in PR#80.

H-02 | Inflation Attack In IPRWAStaking

Category	Severity	Location	Status
Frontrunning	High	IPRWAStaking.sol	Partially Resolved

Description PoC

The `IPRWAStaking` contract is vulnerable to a share inflation attack, which allows an attacker to manipulate the share-to-token exchange rate and extract a disproportionate share of the staked tokens for first stake. This is particularly severe considering there would be new staking contract each IP, creating this situation of first deposit likely on regular basis.

Attack Scenario: Step 1: Attacker Becomes the First Depositor

The attacker ensures they are the first to stake:

- Contract initial state

Step 2: Attacker Manipulates the Exchange Rate

The attacker artificially inflates the ratio by donating tokens directly to the contract:

- New state
- New exchange rate

Step 3: Victim Gets Massively Diluted

Now, a victim stakes a large amount, say 100 IPRWA:

- `stIPRWA` minted to victim
- The victim receives 0 `stIPRWA` for staking 100 IPRWA, due to rounding down. No checks are in place to prevent zero minting.

Step 4: Attacker Unstakes and Profits

The attacker now unstakes their 1 wei `stIPRWA`:

- Unstaked amount:
- Attacker receives almost 1100 IPRWA, effectively stealing the victim's entire deposit.

Additionally, any IPRWA deposit to the staking contract when `stIPRWA` supply is zero, either maliciously (donations) or inadvertently (depositing royalties), will cause the numerator of `_stIPRWAPerIPRWA` calculation to be 0, while the denominator is non zero. Therefore, any subsequent stakes will mint zero `stIPRWA` tokens.

Recommendation

To defend against this class of attacks:

- **Option 1:** Consider performing an initial seed deposit during or immediately after deployment to set a minimum viable exchange rate.
- **Option 2:** Consider using a **virtual offset** as proposed in this ERC-4626 article to smooth out early ratio manipulation.

Resolution

Aria Team: The issue was resolved in PR#100.

H-03 | Blacklist Bypass Via Transfer

Category	Severity	Location	Status
Validation	High	IPRWAStaking.sol: 155-156	Resolved

Description

Aria implements a blacklist mechanism to prevent malicious actors from staking `IPRWA` or unstaking `sIPRWA`. However, this check is only enforced at the staking/unstaking functions and not at the token level.

Since the `IPRWA` and `sIPRWA` token contracts themselves do not enforce blacklist restrictions on transfers, a blacklisted user can simply transfer their tokens to another address to bypass these controls.

Example:
If Alice is blacklisted, she can transfer her `sIPRWA` tokens to another wallet she controls and still successfully unstake from that new address.

Recommendation

Extend the blacklist logic to token transfers by introducing a `beforeTransfer` hook that checks the blacklist for both sender and receiver addresses.

Resolution

Aria Team: The issue was resolved in PR#80.

H-04 | Stake() Susceptible To Griefing

Category	Severity	Location	Status
DoS	High	IPRWAStaking.sol: 209	Resolved

Description

In the stake() function, Aria calculates _stIPRWAPerIPRWA using the following logic:

```
UD60x18 numerator = scaledStakedIPRWATotalSupply.sub(scaledStakedIPRWABalance);
```

This formula subtracts the contract's own stIPRWA balance from the total supply, presumably to account for protocol-held stIPRWA used in royalty distributions.

However, this opens up a griefing vector:

• An attacker can stake IPRWA to receive stIPRWA, then donate those tokens to the staking contract's own address.

• This artificially inflates the scaledStakedIPRWABalance, eventually making it equal to the total supply.

• When this happens, the numerator becomes zero, causing _stIPRWAPerIPRWA to evaluate to zero.

• As a result, subsequent users attempting to stake will receive zero stIPRWA, effectively locking their IPRWA with no yield or liquidity — bricking the vault.

While Aria's rationale for this setup is to allow royalty distributions via protocol-held stIPRWA, this mechanism is problematic. Once stIPRWA is sent to the contract address, it cannot re-enter circulation cleanly:

• Theoretically, an admin could call withdraw() to recover the stIPRWA, but this would reduce the per-token rate, effectively penalizing existing stakers.

• Thus, the system introduces irreversible accounting imbalances and risks permanent vault malfunction.

Recommendation

Avoid factoring in scaledStakedIPRWABalance when computing ratios in both the stake() and unstake() functions. Consider not using sIPRWA for royalties.

Resolution

Aria Team: The issue was resolved in PR#81.

M-01 | Incorrect Fractional Token Supply Check

Category	Severity	Location	Status
Validation	Medium	VaultAssetRegistryAdmin.sol	Resolved

Description

The `_deployFractionalToken` function compares `fractionalTokenTotalSupply` with the `scaledTotalDeposits` value before deploying the fractional token.

However, the `_getScaledTotalDeposits` function only includes deposits associated with the latest USDC address and excludes previous deposits if there is more than one USDC address, causing the check to be performed with incorrect values.

Recommendation

Use the entire deposit amount by iterating over all USDC addresses, similar to how the `_fundraiseCalculateClaim` function does in the `VaultFundraise` contract.

Resolution

Aria Team: Resolved

M-02 | Faulty Pause Role Logic

Category	Severity	Location	Status
Access Control	Medium	IPRWAStaking.sol: 269-270	Resolved

Description

Aria intends to allow either `PAUSABLE_ROLE` holders or `DEFAULT_ADMIN_ROLE` holders to modify the pause state of the protocol.

However, due to the use of logical OR with negated conditions, the current implementation allows only those who hold both roles to execute the function successfully.

Problematic line

```
if (hasRole(PAUSABLE_ROLE, msg.sender) || hasRole(DEFAULT_ADMIN_ROLE, msg.sender)) revert Unauthorized();
```

Example

If Alice holds only `PAUSABLE_ROLE`, then:

- `hasRole(PAUSABLE_ROLE, msg.sender)` → false
- `hasRole(DEFAULT_ADMIN_ROLE, msg.sender)` → true
- The OR condition (false || true) → true → reverts

Recommendation

Update the logic to check that the caller holds at least one of the roles correctly.

Resolution

Aria Team: The issue was resolved in PR#76.

M-03 | Ineffective License Enforcement

Category	Severity	Location	Status
Validation	Medium	License.sol: 13-14	Resolved

Description

Aria introduces a license-signing mechanism that allows users to sign licenses via `signLicense()`. Additionally, admins can revoke licenses from users by calling `revokeSignature()`. However, this system has enforcement gaps:

• Reuse of Signatures: A user whose license has been revoked can still reuse their old signature to sign the license and unstake tokens, as there's no mechanism invalidating the original signed payload.

• Bypassing License Enforcement via Transfer: Even if a user's license is revoked, they can transfer their `sIPRWA` to another address (e.g., via a swap in a public pool like `sIPRWA:IPRWA`) because the `beforeTransfer` hook of `sIPRWA` does not enforce any license-related checks. This allows the receiver—who may not have signed the current license—to exit without needing to sign again.

• License Upgrades Ignored: If an IPRWA issuer upgrades the license (changes the license version or URI), existing `sIPRWA` holders who signed the older license can bypass the need to re-sign the updated license. They can simply swap their tokens in public pools to exit, which again defeats the license enforcement.

Recommendation

• Invalidate Old Signatures: Implement nonce-based replay protection to invalidate previously signed payloads after a license is revoked.

• Enforce License Validation on Transfer: Add a `beforeTransfer` hook in the `sIPRWA` token contract that verifies whether the sender and receiver have signed the current license version. Note that if this is done, it would cause reverts with DeFi pools, since they do not support signing, in that case a explicit skip list for validations would need to be maintained by admin.

Resolution

Aria Team: The issue was resolved in PR#83.

M-04 | Permissionless Vault Creation

Category	Severity	Location	Status
Validation	Medium	AriaIPRWAVaultFactory.sol	Resolved

Description

The `deployFundraiseIpVault` and `deployWhitelistIpVault` functions in the factory contract are permissionless and can result in illegitimate vaults being presented as Aria vaults, potentially tricking users through malicious deployments.

Additionally, whitelist vault initializations require a Merkle root, and generating this root requires the vault address. This means deployers must calculate the address of the vault to be deployed.

However, since vault creation is permissionless, frontrunning and creating another vault from factory changes the predicted address of the whitelist vault. As a result, the Merkle root becomes invalid.

Recommendation

Restrict vault deployments to admins only.

Resolution

Aria Team: The issue was resolved in PR#82.

M-05 | Incorrect Tracking Of usersNetIPRWADeposited

Category	Severity	Location	Status
DoS	Medium	IPRWAStaking.sol: 163	Resolved

Description PoC

The staking contract tracks deposited amounts using the public `usersNetIPRWADeposited` variable. This value increases when users stake and decreases proportionally when they unstake. However, it does not accurately reflect the actual staked amounts, and it can be manipulated.

When the asset-to-share ratio is not 1:1, a user can unstake and then immediately stake again to increase the `usersNetIPRWADeposited` value, since unstaking removes less principal while staking adds the full amount.

Currently, that value is not used beyond being a public variable. However, the impact could be more significant if Aria itself, or other protocols integrating with the Aria staking contracts, rely on `usersNetIPRWADeposited` for their operations.

Recommendation

Be aware of this behavior and document it. Alternatively, consider removing the `usersNetIPRWADeposited` variable if it is not planned to be used.

Resolution

Aria Team: The issue was resolved in PR#99.

M-06 | Sandwich Attack Royalties

Category	Severity	Location	Status
Gaming	Medium	IPRWAStaking.sol: 162	Partially Resolved

Description PoC

When users unstake their shares from the `IPRWAStaking`, the amount to withdraw is calculated by the `_getIPRWAValue`. This will calculate the `IPRWA:stIPRWA` to convert the `stIPRWA` tokens burned to `IPRWA`.

However, this rate is based on the token balance of the staking contract. This will cause the rate to have a stepwise jump when royalties are deposited.

Malicious users can monitor these royalties deposits, and sandwich the transaction by staking and unstaking, before and after the `depositRoyalties` call, profiting from the royalties that should have been distributed to existing users.

Recommendation

Consider rate based logic or a delayed distribution system, that will prevent royalties to be instantly claimed and avoid step wise jumps. Alternatively, increase the frequency of royalty deposits, to reduce the profit for attackers and make it less likely.

Please be cautious of DEX pools while designing this, as Aria expects to have pools for `sIPRWA/IPRWA` and `IPRWA/USDC`. Depending on how royalties are handled, this could create instant arbitrage opportunities on the DEX.

If you decide to go with a rate-based or delayed royalty distribution model, consider creating a wrapped `sIPRWA` specifically for use in DeFi pools.

Resolution

Aria Team: Partially Resolved.

L-01 | Signature-Based Mapping Vulnerability

Category	Severity	Location	Status
Validation	Low	Licensed.sol: 35-36	Resolved

Description

Aria uses the signature itself as the key in the mapping:

`signatures[msg.sender] = signature;`

However, be aware that for the same digest, multiple valid signatures can exist. This allows a user to replay their signature using the same digest but with different signature bytes.

This bypasses the `License_AlreadySigned` check and emits the `LicenseSigned` event unnecessarily, which could confuse integrators or UIs.

While there is no direct incentive to do this—since the function is gated by msg.sender—we wanted to flag this behavior for awareness.

Recommendation

Consider using a mapping like `signed[userAddress] = digest` instead.This would ensure that new signatures are only accepted when `signed[userAddress] ≠ digest`.

Resolution

Aria Team: The issue was resolved in PR#83.

L-02 | Smart Wallets Cannot Stake

Category	Severity	Location	Status
DoS	Low	Licensed.sol: 23-24	Resolved

Description

Aria's license system requires users to sign a license in order to stake or unstake. However, the current signature verification logic implemented by Aria does not support smart contract signatures (`EIP-1271`).

As a result, smart contract wallets are unable to stake or unstake via the IPRWA staking contract.

Recommendation

Consider supporting smart contract signatures by implementing `EIP-1271` verification.

Resolution

Aria Team: The issue was resolved in PR#98.

L-03 | Lack Of Slippage Controls

Category	Severity	Location	Status
Validation	Low	IPRWAStaking.sol: 137-138	Resolved

Description

The `IPRWAStaking` contract implements stake and unstake functionalities but currently lacks slippage controls.

Slippage controls can be desirable to protect users against inflation-based attacks or to give them more control over the expected return value during staking or unstaking.

Recommendation

Consider adding an optional `minOut` parameter to both stake and unstake functions to allow users to specify a minimum acceptable return and revert the transaction if the actual amount falls below it.

Resolution

Aria Team: The issue was resolved in PR#95.

L-04 | Front-Running And Value Leakage

Category	Severity	Location	Status
Frontrunning	Low	Global	Acknowledged

Description

Aria expects to receive royalties in USDC, which are later converted into IPRWA and deposited into the staking contract as rewards. However, this approach poses a few potential issues:

1. Front-running risk due to predictable swap flow
As we understand, the expected royalty flow is:
• Onramp: Royalties arrive in IP
• Coinbase: IP is swapped to USDC
• Offramp: USDC is transferred to Story (RWAKfeller's ledger)
• Story: USDC is swapped to IPRWA on PiperX

The problem begins from the offramp stage onward, which is publicly observable. Once USDC reaches the Story address, any external actor can anticipate the swap and front-run it by executing a USDC → IPRWA trade in the IPRWA/USDC pool. They profit from the price movement induced by Aria's expected buy, effectively sandwiching the trade. This causes Aria to acquire IPRWA at a worse rate, thereby diluting the yield meant for stakers — a subtle form of MEV extraction.

2. Royalty value leakage to non-stakers
Since the swap is done on the open market (IPRWA/USDC pool), the resulting price impact benefits all IPRWA holders, not just stakers. Thus, a portion of the royalty value leaks to non-staking holders, reducing the effective yield for those who have actually staked.

Recommendation

To mitigate these risks:

- Perform swaps in randomized, smaller tranches without a predictable schedule.
- Use fresh recipient addresses for each offramp to obscure tracking.
- If the royalty amount is large relative to on-chain liquidity, split it across multiple fresh addresses and execute the swaps from them independently.
- For extreme cases with low liquidity and large royalty anticipation, consider introducing a beforeTransfer hook in the IPRWA token contract. Aria can:
    - Temporarily pause transfers by setting a `allowTransfers = false` flag,
    - Enable swaps by toggling `allowTransfers = true` within a dedicated smart contract, execute the swap, and then re-disable transfers.

Resolution

Aria Team: Acknowledged.

L-05 | Arbitrage Risk From Royalty Deposits

Category	Severity	Location	Status
Frontrunning	Low	Global	Acknowledged

Description

Aria expects to maintain a DEX pool with the token pair `sIPRWA/IPRWA`. However, each time royalties are deposited into the staking contract, the vault value backing `sIPRWA` increases relative to `IPRWA`.

This creates an arbitrage opportunity in the DEX pool:

The price of `sIPRWA` lags behind its new intrinsic value post-deposit, allowing external actors to profit from the mismatch — potentially at the expense of the protocol or its users.

Recommendation

Be aware of this arbitrage vector.
Since Aria controls the royalty deposit process, it can easily capture this value internally using a atomic multicall:

1. Swap USDC to IPRWA

2. Deposit IPRWA as royalties into the staking contract

3. Swap IPRWA to `sIPRWA` in the DEX pool, to reflect the updated vault value

This approach prevents external actors from frontrunning the arbitrage opportunity and ensures the value created from royalty

Resolution

Aria Team: Acknowledged.

L-06 | Claims DoS'ed By Admin Fractional Minting

Category	Severity	Location	Status
Validation	Low	VaultAdmin.sol: 127	Resolved

Description PoC

Fractional tokens are based on `ERC20CappedUpgradeable`, and the cap is set to `fractionalTokenTotalSupply` at the time of registration in the `_deployFractionalToken` function.

After an IP is fractionalized, users can claim their fractional tokens in proportion to their individual deposit amounts relative to the total deposits. The cap, defined by `fractionalTokenTotalSupply`, is reached once all users have claimed their tokens.

There is an additional fractional token minting mechanism initiated by the admin, which can be executed by anyone after a timelock period. This process involves the `initFractionalTokenMint` and `execFractionalTokenMint` functions and allows the remaining tokens to be minted to a specified address.

However, the `initFractionalTokenMint` function does not include a check to ensure that the `claimDeadline` has passed and that all user claims have been completed.

The `initFractionalTokenMint` function can be called while user claims are still ongoing, and the `_getAdjustedMintAmount` function incorrectly returns a higher mintable amount, as it only compares the current supply against the cap.

This results in later users being unable to claim their fractional tokens due to the cap being reached, effectively causing them to lose their funds. Additionally, although the function is restricted to the admin, the admin does not have to be malicious and may simply be unaware of this behavior.

Recommendation

The `initFractionalTokenMint` function should check the `claimDeadline` and only be callable after user claims have been completed.

Resolution

Aria Team: The issue was resolved in PR#91.

L-07 | Warning Regarding Story Registration Fee

Category	Severity	Location	Status
Warning	Low	VaultAssetRegistryAdmin.sol: 244	Resolved

Description

During the registration and fractionalization process, the `mintAndRegisterIpAndAttachPILTermsAndDistributeRoyaltyTokens` function of the `RoyaltyTokenDistributionWorkflows` periphery contract is called.

This contract handles the registration of the IP asset on the Story chain. The `RoyaltyTokenDistributionWorkflows` contract calls the `IPRegistry.register` function, and the `registerFeePayer` is msg.sender during this call.

This means the periphery contract is responsible for paying the registration fee if `feeAmount` is greater than zero. There isn't an issue at the moment, as the registration fee is set to zero on live contracts.

However, if the Story Protocol decides to increase the fee, the `RoyaltyTokenDistributionWorkflows` contract will be responsible for paying it. This would likely require the fee tokens to be transferred from the Aria vault to the periphery contract during the registration process.

Recommendation

Be aware of this situation. Since Aria vaults are also upgradeable, they can be modified to align with changes in the Story Protocol's behavior.

Resolution

Aria Team: The issue was resolved in PR#86.

L-08 | Missing IPAccount Execute Functionality

Category	Severity	Location	Status
Informational	Low	Global	Resolved

Description

The IP asset NFT is held in the vault, and each IP asset has an associated IP account. These IP accounts are bound to the IP asset, and the NFT owner is also the owner of the corresponding IP account.

IP accounts need to be utilized, using their `execute` and `executeWithSig` functions, in order to interact with the Story Protocol's modules. However, the vault has no way to interact with its own IP account.

Recommendation

Consider implementing a similar execute function that allows the vault to interact with its IP account in order to fully utilize all features of the Story Protocol. Alternatively, if this limitation is intentional, it should be clearly documented.

Resolution

Aria Team: The issue was resolved in PR#96.

L-09 | Fundraise Closed With No Expiration Time Check

Category	Severity	Location	Status
Validation	Low	VaultFundraiseAdmin.sol: 44	Resolved

Description

The `VaultStateChecker` library is in charge of updating the vault state to `Closed` if the expiration time is reached. However, admin can close the fundraise without checking this timestamp, creating inconsistencies between the library.

Recommendation

If this is intended behavior, consider documenting it for users, so they are aware admin can close the fundraise at any time. Alternatively, validate if the expiration time is reached in `closeRaise` admin function.

Resolution

Aria Team: The issue was resolved in PR#90.

L-10 | Failed Withdrawals Due To Previous Withdraw

Category	Severity	Location	Status
Informational	Low	VaultFundraiseAdmin.sol: 96	Resolved

Description

The vault contract includes both `withdraw()` and `withdraw(address token)` functions. The `withdraw()` function iterates through all USDC token addresses and transfers the entire deposited amounts. The `withdraw(address token)` function transfers the deposited amount for the specified token.

However, if the `withdraw(address token)` function is called once, the regular `withdraw()` function can no longer be used for the remaining USDC addresses. This is because the iteration will revert due to insufficient balance when it reaches the token address that was already withdrawn.

Recommendation

Document this behavior for vault admins. They should either use the regular `withdraw` function exclusively, or call the `withdraw(address token)` function separately for each token address.

Resolution

Aria Team: The issue was resolved in PR#103.

L-11 | Merkle Root Updates After Claim

Category	Severity	Location	Status
Logical Error	Low	VaultWhitelistAdmin.sol: 33	Resolved

Description

Whitelisted users can claim their fractionalized tokens using Merkle proofs, and each user can only claim once. However, the vault admin can update the Merkle root at any time, regardless of whether the previous root was used for claims.

If a user has already claimed their tokens with the old root but is allocated a higher amount in the new root, they will be unable to claim the difference because `fractionalTokenClaimed` remains true for their account, resulting in a loss of tokens.

Conversely, if a user claimed based on the previous root but has a much lower allocation in the new root, this could cause the overall cap to be reached unexpectedly.

Recommendation

One option to consider is setting a claim start time, allowing the admin to verify everything is correct before claims begin and preventing Merkle root updates once claiming has started. Alternatively add a check in the `setMerkleRoot` and allow only if the token has not been fractionalized yet.

Resolution

Aria Team: The issue was resolved in PR#89.

L-12 | Whitelist Claims DoS'ed

Category	Severity	Location	Status
Configuration	Low	AriaIPRWAVault.sol: 105	Resolved

Description

During `claimFractionalTokens` for whitelist vaults, the same `_USDC_WHITELIST` address will be used and `fractionalTokenClaimed[usdc][claimer]` will be set to true. In case the same claimer address is present in different leaves, the user will only be able to claim one time.

Recommendation

Consider verifying the merkle tree does not contain more than one leaf with the same user address.

Resolution

Aria Team: The issue was resolved in PR#88.

L-13 | recoverLostTokens Blocked By Stale totalDeposits

Category	Severity	Location	Status
Logical Error	Low	VaultAdmin.sol: 189-192	Resolved

Description

The `recoverLostTokens` function is prevented from withdrawing the deposited USDC tokens from users.

However if the vault closed and the admin withdraws all of these deposited USDC tokens the `totalDeposits` variable is not reset to 0 as it is later used for the share price calculation in the claim flow. This blocks the `recoverLostTokens` function from legitimate actions.

Recommendation

Do not decrease the total deposited USDC value from the recoverable amount if the admin already withdraw the funds.

Resolution

Aria Team: The issue was resolved in PR#103.

L-14 | Blacklisted Tokens Permanently Locked

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

The protocol has a blacklist feature, however once tokens are blacklisted there is no way for the admin to seize those assets, effectively locking them in perpetuity.

Recommendation

Consider adding an option for the admin to seize blacklisted assets.

Resolution

Aria Team: The issue was resolved in PR#80.

I-01 | Unused Code

Category	Severity	Location	Status
Superfluous Code	Info	Global	Resolved

Description

The error `IPRWAStaking.IncorrectSignatureLength` is never used.

Recommendation

Remove unused error.

Resolution

Aria Team: The issue was resolved in PR#93.

I-02 | Incorrect Natspec

Category	Severity	Location	Status
Documentation	Info	Global	Resolved

Description

- `IAriaIPRWAVaultFactory.deployFundraiseIpVault` and `IAriaIPRWAVaultFactory.deployWhitelistIpVault` have a `withdrawalTimelockDuration` but this refers to a previous feature.
- `VaultFundraiseAdmin.withdraw()` missing natspec.

Recommendation

Fix the natspec for the function above.

Resolution

Aria Team: The issue was resolved in PR#92.

I-03 | Whitelist Vault Claims Can Fail

Category	Severity	Location	Status
Configuration	Info	AriaIPRWAVault.sol: 123	Resolved

Description

Extra attention should be made before calling the `registerIPAndFractionalize` on a whitelist vault, as once the fractional token is deployed, the total supply is fixed.

If there is any inconsistency with the total claim amounts in the merkle tree, users will be DoS'ed when claiming their IPRWA tokens.

This is different from the fundraise vault, as the token deployment will fail if the supply is not above the scaled total deposits.

Recommendation

Add a warning documentation to the `registerIPAndFractionalize` to verify the total amounts to claim in merkle tree before deploying the fractional token.

Resolution

Aria Team: The issue was resolved in PR#94.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits