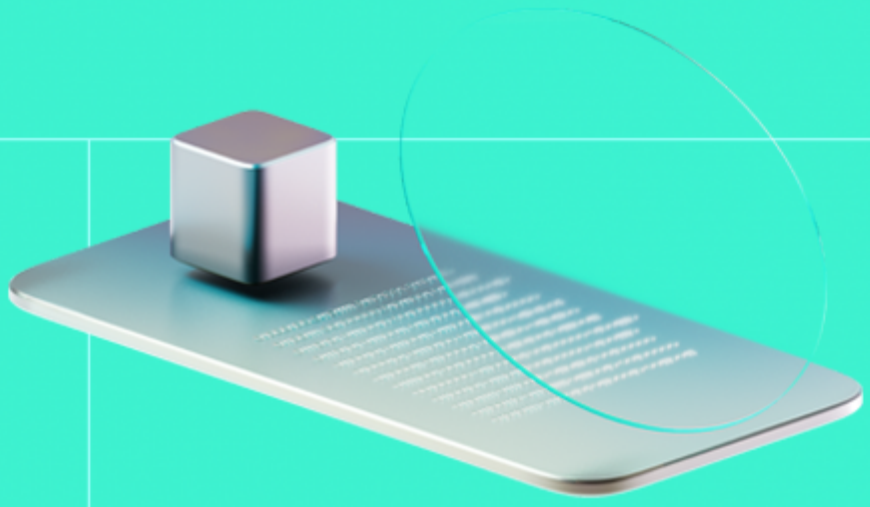
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Upshift Finance

Date: 30/01/2026

We express our gratitude to the Upshift Finance team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Upshift Vault is a core ERC-4626 vault that enables users to deposit funds while earning yield through deployment to August subaccounts, which manage strategies securely across multiple chains. It simplifies user experience with a single reference token, supports multi-chain DeFi opportunities, and enforces strict roles and permissions for secure capital management.

Document

Name	Smart Contract Code Review and Security Analysis Report for Upshift Finance
Audited By	David Camps Novi
Approved By	Turgay Arda Usman
Website	https://www.upshift.finance/
Changelog	30/01/2026 - Preliminary Report; 30/01/2026 - Final Report;
Platform	Any EVM-compatible chain
Language	Solidity
Tags	ERC4626; Upgradable; Yield Farming; Centralization; Claims; Vault
Methodology	https://docs.hacken.io/methodologies/smart-contracts

Review

Scope

Repository	https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist
Initial Commit	4a08f62
Final Commit	78792e4

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

3	2	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	0
Low	0

Vulnerability	Severity	Status
F-2026-14864 - Floating Pragma	Info	Accepted
F-2026-14887 - Unused Code	Info	Fixed
F-2026-14888 - Missing Zero Address Check for Contract Owner	Info	Fixed

Documentation quality

- Functional requirements are limited.
- Technical description is limited.

Code quality

- The development environment is well-configured.
- Code architecture has a modular design with clear separation of concerns.
- NatSpec is present but does not extensively describe functionality.

Test coverage

Code coverage of the project is **29.00%** (branch coverage)

- Branch coverage from `SendersAllocationWhitelist.sol` is **84.00%**. Remaining contracts' tests fail due to configuration errors.
- Deployment and basic user interactions are not covered with tests.
- Negative cases coverage is missed.
- Interactions by several users are not tested thoroughly.

Table of Contents

System Overview

Upshift Vault is a core ERC-4626 vault that enables users to deposit funds while earning yield through deployment to August subaccounts, which manage strategies securely across multiple chains. It simplifies user experience with a single reference token, supports multi-chain DeFi opportunities, and enforces strict roles and permissions for secure capital management.

The system consists of the following contracts:

- **TimelockedVault** - Adds withdrawal timelocks and instant redemption fee logic to the vault, enforcing delayed withdrawals and tracking lag durations for security and proper fee application.
- **OraclizedMultiAssetVault** - Manage deposits, withdrawals, and subaccount interactions across multiple whitelisted assets.
- **SendersAllocationWhitelist** - Manages the whitelist of vault users with a dynamic allocation system.

Privileged roles

- **Owner**
  - Manages whitelisted allocations.
  - Ability to modify deposit and withdraw limits.
  - Sets system fees.
  - Defines the whitelisted assets of the vaults.
  - Capacity to manage the deposits and withdrawals to subAccounts for yield administration.
  - Can update the value of the total assets in the vaults.
- **Operator**
  - Capacity to manage the deposits and withdrawals to subAccounts for yield administration.
  - Can update the value of the total assets in the vaults.

Potential Risks

- The owner of the `SendersAllocationWhitelist` contract can alter user participation through the privileged functions `setAllocation()` and `disableSender()`. This allows the owner to modify user allocations both before and after engagement, effectively enabling unilateral control over users' ability to participate. As a result, users must place full trust in the contract owner, as misuse or compromise of the owner role could directly impact user allocations and participation rights.
- The owner of the `SendersAllocationWhitelist` contract can modify users' used allocation values via the `setDeposited` function. This grants the owner the ability to arbitrarily alter user state after deposits have occurred, requiring users to fully trust the owner, as misuse or compromise of this role could directly impact system integrity and fairness.

Findings

Vulnerability Details

F-2026-14864 - Floating Pragma - Info

Description:

In Solidity development, the pragma directive specifies the compiler version to be used, ensuring consistent compilation and reducing the risk of issues caused by version changes. However, using a floating pragma (e.g., `^0.8.xx`) introduces uncertainty, as it allows contracts to be compiled with any version within a specified range. This can result in discrepancies between the compiler used in testing and the one used in deployment, increasing the likelihood of vulnerabilities or unexpected behavior due to changes in compiler versions.

The project currently uses floating pragma declarations (`^0.8.20`) in its Solidity contracts. This increases the risk of deploying with a compiler version different from the one tested, potentially reintroducing known bugs from older versions or causing unexpected behavior with newer versions. These inconsistencies could result in security vulnerabilities, system instability, or financial loss. Locking the pragma version to a specific, tested version is essential to prevent these risks and ensure consistent contract behavior.

Assets:

- src/core/SendersAllocationWhitelist.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]
- src/tokenized-vaults/base/OraclizedMultiAssetVault.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]
- src/tokenized-vaults/base/TimelockedVault.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]

Status: Accepted

Classification

Impact Rate: 2/5

Likelihood Rate: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Info

Recommendations

Remediation: It is recommended to **lock the pragma version** to the specific version that was used during development and testing. This ensures that the contract will always be compiled with a known, stable compiler version, preventing unexpected changes in behavior due to compiler updates. For example, instead of using `^0.8.xx`, explicitly define the version with `pragma solidity 0.8.19;`.

Before selecting a version, review known bugs and vulnerabilities associated with each Solidity compiler release. This can be done by referencing the official Solidity compiler release notes: Solidity GitHub releases or Solidity Bugs by Version. Choose a compiler version with a good track record for stability and security.

Resolution: The Upshift team Accepted this finding, providing the following feedback.

> The pragma defines the minimum supported constraint. Compiling the contract with the right version is our responsibility.

F-2026-14887 - Unused Code - Info

Description: The contract `OraclizedMultiAssetVault` declares the following state variable:

```
/// @dev Maximum basis points (100%)
uint256 internal constant MAX_BPS = 10_000;
```

However, this variable is not used within the context of this contract, decreasing code quality and readability.

Assets:

- src/tokenized-vaults/base/OraclizedMultiAssetVault.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]

Status: Fixed

Classification

Impact Rate: 1/5

Likelihood Rate: 5/5

Exploitability: Independent

Complexity: Simple

Severity: Info

Recommendations

Remediation: It is recommended to remove unused code.

Resolution: Fixed in commit ID `78792e4`: the unused code was removed from the reported contract.

F-2026-14888 - Missing Zero Address Check for Contract Owner - Info

Description: The constructor of `SendersAllocationWhitelist` accepts an `ownerAddr` parameter but does not validate that it is not the zero address (`address(0)`).

```
constructor(address ownerAddr) {
    _owner = ownerAddr;
}
```

The contract relies on the `_owner` address as an access control mechanism for multiple administrative functions. If `_owner` is wrongly set to the zero address during deployment, all owner-restricted functions will become permanently inaccessible. Since `SendersAllocationWhitelist` depends on the owner role to manage and configure core functionality, this misconfiguration could render the contract unusable, leaving no way to recover control without redeploying the contract.

Assets:

- src/core/SendersAllocationWhitelist.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]

Status: Fixed

Classification

Impact Rate: 3/5

Likelihood Rate: 1/5

Exploitability: Dependent

Complexity: Simple

Severity: Info

Recommendations

Remediation: Consider adding a zero address check in the constructor to ensure a valid owner is provided during deployment.

Resolution: Fixed in commit ID `78792e4`: a zero address check was added into the reported method.

```
constructor(address ownerAddr) {
    if (ownerAddr == address(0)) revert InvalidAddress();
    _owner = ownerAddr;
}
```

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

As part of Hacken's ongoing quality assurance process, we may conduct re-audits of select projects. These re-audits are performed independently from the original audit and are intended solely for internal quality control and improvement. Updated reports resulting from such re-audits will be shared privately with the respective clients and may be published on the Hacken website only with their explicit consent.
The sole authoritative source for finalized and most up-to-date versions of all reports remains the Audits section at https://hacken.io/audits/.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project's performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist
Initial Commit	4a08f62
Final Commit	78792e4
Whitepaper	N/A
Requirements	https://docs.upshift.finance/architecture/vault-architecture
Technical Requirements	https://docs.upshift.finance/architecture/vault-architecture

Asset	Type
src/core/SendersAllocationWhitelist.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]	Smart Contract
src/tokenized-vaults/base/OraclizedMultiAssetVault.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]	Smart Contract
src/tokenized-vaults/base/TimelockedVault.sol [https://github.com/fractal-protocol/august-contracts-v2/tree/allocation-whitelist/src]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.

Frameworks and Methodologies

This security assessment was conducted in alignment with recognised penetration testing standards, methodologies and guidelines, including the NIST SP 800-115 – Technical Guide to Information Security Testing and Assessment, and the Penetration Testing Execution Standard (PTES), These assets provide a structured foundation for planning, executing, and documenting technical evaluations such as vulnerability assessments, exploitation activities, and security code reviews. Hacken's internal penetration testing methodology extends these principles to Web2 and Web3 environments to ensure consistency, repeatability, and verifiable outcomes.