Unitas Audit

Audit Report

Fri Sep 12 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Unitas Audit Audit Report

1 Executive Summary

1.1 Project Information

Description	Unitas is Unipay's on-chain "dollar + yield" stack, built first and foremost on a JLP delta-neutral (dn) arbitrage engine.
Type	Yield Aggregator
Auditors	ScaleBit
Timeline	Mon Sep 01 2025 - Thu Sep 04 2025
Languages	Solidity
Platform	Multichain
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/UnipayFI/unitas-evm-contract
Commits	4290d4489e14ea167242d172c5b110dacfda84d5

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
USD1	contracts/USDu.sol	53e948dddd2e7fc8cbc88210d115fafa87c3d1f3
SAAC	contracts/SingleAdminAccessControl.sol	5b2975b41ca556d22faf52ed7426ad0f73afd190
UMI1	contracts/UnitasMinting.sol	46854ba3711f2573876af1f6bf99af0cd218b447
UMV2	contracts/UnitasMintingV2.sol	d1196a242ed9aecedbe58ded77ad4b157e509a68
SUSD	contracts/StakedUSDu.sol	bde5c2cac0653cc3580924a04100fe8ed1c1c776
USDS	contracts/USDuSilo.sol	03e2110362e9e340e7a995ec12ebd4a4303a5d5b
SUSDV2	contracts/StakedUSDuV2.sol	6a38bd5f82652322a93685f2f2ba3d6cf602e687

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	13	5	8
Informational	0	0	0
Minor	4	1	3
Medium	4	2	2
Major	5	2	3
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Unitas to identify any potential issues and vulnerabilities in the source code of the unitas-evm-contract smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 13 issues of varying severity, listed below.

ID	Title	Severity	Status
SUS-1	FULL_RESTRICTED_STAKER_ROLE Blacklist can be Bypassed via ERC4626 `allowance/permit`	Major	Fixed
SUS-2	Cooldown Griefing: Authorized Spenders can Keep Extending the Lock	Major	Acknowledged
SUS-3	Unstake Ignores FULL_RESTRICTED_STAKER_ROLE after Cooldown	Medium	Fixed
UMI-1	Mint and Redeem Can Be Exploited as a Lossless Arbitrage	Major	Acknowledged
UMI-2	USDu int/redeem Relies Fully on off-chain Price and Signatures	Medium	Acknowledged
UMI-3	Inconsistent Asset Flow Between `mint` and `redeem`	Minor	Acknowledged

UMI-4	Nonce Handling Uses only the Lower 64 Bits	Minor	Acknowledged
SUS1-1	FULL_RESTRICTED_STAKER_ROLE Address can still Deposit and Mint Shares to Someone Else	Major	Fixed
SUS1-2	SOFT_RESTRICTED_STAKER_ROLE Restriction does not Cover Redemption Paths	Major	Acknowledged
SUS1-3	Centralization Risks	Medium	Acknowledged
SUS1-4	Donation Attack not Fully Protected	Medium	Fixed
SUS1-5	Docs and Code Mismatch for Blacklist Administration	Minor	Fixed
SUS1-6	Redundant Unvested Amount Calculation	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the unitas-evm-contract Smart Contract :

Admin Roles

- The `REWARDER_ROLE` can transfer rewards from the controller contract into this contract through `transferInRewards()` .
- The `BLACKLIST_MANAGER_ROLE` can add/remove an address through `addToBlacklist()` / `removeFromBlacklist()` .
- The `DEFAULT_ADMIN_ROLE` can burn the full restricted user amount and mint to the desired owner address through `redistributeLockedAmount()` .
- The `DEFAULT_ADMIN_ROLE` can transfer tokens from smart contracts except the `asset()` through `rescueTokens()` .
- The `DEFAULT_ADMIN_ROLE` can set cooldown duration through `setCooldownDuration()` .
- The owner can update the minter through `setMinter()` .
- The minter can mint `USDu` througn `mint()` .
- The `MINTER_ROLE` / `REDEEMER_ROLE` can mint/redeem stablecoins from assets through `mint()` / `redeem()` .
- The `DEFAULT_ADMIN_ROLE` can update the `maxMint` / `maxRedeem` through `setMaxMintPerBlock()` / `setMaxRedeemPerBlock()` .
- The `GATEKEEPER_ROLE` can disable the mint and redeem through `disableMintRedeem()` .
- The `MINTER_ROLE` can transfer an asset to a custody wallet through `transferToCustody()` .
- The `DEFAULT_ADMIN_ROLE` can add/remove an asset to the supported assets list through `addSupportedAsset()` / `removeSupportedAsset()` .
- Other privileged functions.

Users

- Users can mint/withdraw/redeem/unstake `USDu` through `mint()` / `withdraw()` / `redeem()` / `unstake()`.

4 Findings

SUS-1 FULL_RESTRICTED_STAKER_ROLE Blacklist can be Bypassed via ERC4626 `allowance/permit`

Severity: Major

Status: Fixed

Code Location:

contracts/StakedUSDuV2.sol#95-106,111-122;

contracts/StakedUSDu.sol#224-237

Descriptions:

In the vault's ERC4626 flow, `_withdraw(caller, receiver, owner, ...)` checks `FULL_RESTRICTED_STAKER_ROLE` only for `caller` and `receiver`, not for `owner`. Under ERC4626, a spender with allowance/permit can withdraw on the owner's behalf; because `owner` isn't checked for `FULL_RESTRICTED_STAKER_ROLE`, withdrawals can burn the owner's shares and deliver USDu to a non-blacklisted receiver. In V2's cooldown flow, the same unchecked owner flows into `cooldownAssets` / `cooldownShares`, and later `unstake` has no FULL guard. As a result, a blacklisted account's funds are not actually frozen. A non-blacklisted helper can drain via withdraw (V1) or start cooldown (V2).

Suggestion:

It is advised to check `owner` for `FULL_RESTRICTED_STAKER_ROLE` (and `SOFT_RESTRICTED_STAKER_ROLE`) in `_withdraw` and cooldown.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SUS-2 Cooldown Griefing: Authorized Spenders can Keep Extending the Lock

Severity: Major

Status: Acknowledged

Code Location:

contracts/StakedUSDuV2.sol#95,111

Descriptions:

Each `cooldownAssets` / `cooldownShares` call resets `cooldownEnd` to now + duration and increases the pending amount. Because the function accepts an `owner` parameter and withdraws using the ERC-4626 `_withdraw` path, any spender with the owner's allowance can repeatedly call it to push out the unlock time. As a result, a malicious authorized party can indefinitely delay the owner’s ability to unstake.

Suggestion:

Allow only the owner to initiate cooldown.

SUS-3 Unstake Ignores FULL_RESTRICTED_STAKER_ROLE after Cooldown

Severity: Medium

Status: Fixed

Code Location:

contracts/StakedUSDuV2.sol#78

Descriptions:

In `StakedUSDuV2`, `unstake(receiver)` has no `FULL_RESTRICTED_STAKER_ROLE` (or `SOFT_RESTRICTED_STAKER_ROLE`) checks, which means a user can be placed on blacklisted/restricted roles yet still withdraw from the silo after cooldown, defeating the freeze objective.

Suggestion:

Block `unstake` for blacklisted/restricted callers/receivers.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

UMI-1 Mint and Redeem Can Be Exploited as a Lossless Arbitrage

Severity: Major

Status: Acknowledged

Code Location:

contracts/UnitasMinting.sol#161,193

Descriptions:

The contract's mint and redeem functions can be executed by any account holding the MINTER_ROLE or REDEEMER_ROLE. Orders are signed by a benefactor and validated on-chain via EIP-712 signatures, but nothing prevents the same participant from repeatedly signing multiple mint orders in advance.

This creates the following arbitrage vector:

- Pre-sign multiple mint orders when the exchange rate between USDu and collateral assets is favorable.
- Wait for asset prices to change in a way that makes redemption more profitable.
- Execute the previously signed mint orders, immediately followed by redeem orders, extracting the spread.

Because both mint and redeem are role-gated but do not enforce any slippage, oracle pricing, or minimum return guarantees, the attacker can always revert or refrain from calling in unfavorable situations, but fully exploit favorable market movements. This results in risk-free (lossless) arbitrage against the system.

Suggestion:

Instead of allowing static signed orders, calculate the USDu collateral conversion rate at execution time using an **oracle** or **on-chain AMM reference price**.

UMI-2 USDu int/redeem Relies Fully on off-chain Price and Signatures

Severity: Medium

Status: Acknowledged

Code Location:

contracts/UnitasMinting.sol#161,193,338

Descriptions:

`verifyOrder` verifies EIP-712 signature/expiry and basic fields, but applies no on-chain price sanity checks. `mint` and `redeem` move collateral/USDu strictly per the signed order amounts. Delayed/wrong off-chain prices enable depeg arbitrage (buy discounted USDu and redeem 1:1 collateral) that drains liquidity and worsens peg instability.

Suggestion:

Add on-chain price bounds/rate limit checks.

UMI-3 Inconsistent Asset Flow Between `mint` and `redeem`

Severity: Minor

Status: Acknowledged

Code Location:

contracts/UnitasMinting.sol#161,193

Descriptions:

The asset transfer logic between `mint` and `redeem` is inconsistent. In the `mint` function, user-supplied assets are transferred from the benefactor to **multiple external addresses**, rather than being retained in the vault contract. This means the vault does not actually custody the full amount of assets corresponding to the minted shares. In the `redeem` function, assets are transferred out **from the vault contract itself**, assuming that the collateral remains available on-chain. This asymmetry creates a structural mismatch: minted shares may not be fully backed, since the vault does not hold the assets but still issues redeemable shares. When a redeem occurs, the vault contract may not have sufficient assets on hand to honor withdrawals, leading to potential insolvency.

Suggestion:

On `redeem`, release assets from the external addresses back to the redeemer.

UMI-4 Nonce Handling Uses only the Lower 64 Bits

Severity: Minor

Status: Acknowledged

Code Location:

contracts/UnitasMinting.sol#376;

contracts/UnitasMintingV2.sol#559

Descriptions:

The nonce bitmap derives `slot = uint64(nonce) >> 8; bit = 1 << uint8(nonce)`, ignoring the upper 192 bits (and upper 64 if using `uint128` elsewhere), which may mislead integrators expecting a full-width `uint256` / `uint128` nonce.

Suggestion:

Document nonce as `uint64` or extend the bitmap to a wider space.

SUS1-1 FULL_RESTRICTED_STAKER_ROLE Address can still Deposit and Mint Shares to Someone Else

Severity: Major

Status: Fixed

Code Location:

contracts/StakedUSDu.sol#202,243

Descriptions:

`_deposit` only blocks `SOFT_RESTRICTED_STAKER_ROLE` addresses; it does not block a `FULL_RESTRICTED_STAKER_ROLE` caller. The transfer hook blocks mint to `FULL_RESTRICTED_STAKER_ROLE`, but allows a FULL-restricted caller to deposit USDu and mint shares to a non-FULL receiver, laundering value into third-party vault shares.

Suggestion:

Disallow deposits initiated by FULL callers regardless of the receiver.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SUS1-2 SOFT_RESTRICTED_STAKER_ROLE Restriction does not Cover Redemption Paths

Severity: Major

Status: Acknowledged

Code Location:

contracts/StakedUSDu.sol#224;

contracts/StakedUSDuV2.sol#78,95,111

Descriptions:

`SOFT_RESTRICTED_STAKER_ROLE` is enforced on `_deposit` but not on `_withdraw`, cooldown, or `unstake`. A `SOFT_RESTRICTED_STAKER_ROLE`-restricted user can obtain stUSDu off-chain and then redeem USDu normally through the contract.

Suggestion:

Apply `SOFT_RESTRICTED_STAKER_ROLE` checks consistently to withdraw/cooldown/unstake to align behavior with policy.

SUS1-3 Centralization Risks

Severity: Medium

Status: Acknowledged

Code Location:

contracts/StakedUSDu.sol;

contracts/StakedUSDuV2.sol;

contracts/UnitasMinting.sol;

contracts/USDuSilo.sol

Descriptions:

Privileged roles are centralized and, in several places, can make immediate changes affecting minting/redemption limits, asset/custodian lists, blacklisting, and even forced balance redistribution. Compromise or operational error on any of these roles can lead to loss of funds, protocol halt, or governance friction.

In **StakedUSDu**:

- `DEFAULT_ADMIN_ROLE` : Rescue non-underlying tokens; forcibly redistribute/burn balances of FULL-blacklisted addresses; grant/revoke roles.
- `BLACKLIST_MANAGER_ROLE` : Add/remove addresses to soft/full blacklist.
- `REWARDER_ROLE` : Pull reward tokens into the vault and (re)start vesting.

In **StakedUSDuV2**:

- `DEFAULT_ADMIN_ROLE` : Configure cooldown duration for stake withdrawals.

In **UnitasMinting**:

- `MINTER_ROLE` : Execute mint orders; transfer collateral out to custody wallets.
- `REDEEMER_ROLE` : Execute redeem orders (burn USDu, send collateral to beneficiary).
- `GATEKEEPER_ROLE` : Disable mint/redeem globally; strip minter/redeemer roles.

- `DEFAULT_ADMIN_ROLE` : Set per-block mint/redeem caps; add/remove supported assets; add/remove custodian addresses.

In **USDu**:

- `owner` : Set the single minter; minter can mint USDu arbitrarily.

In **USDuSilo**:

- `StakingVault` : Only the designated staking vault may withdraw USDu from the silo to a receiver.

Suggestion:

Move all privileged roles to multisig + timelock and carefully manage all the roles.

SUS1-4 Donation Attack not Fully Protected

Severity: Medium

Status: Fixed

Code Location:

contracts/StakedUSDu.sol#190

Descriptions:

In `ERC4626`, minted shares are computed as `floor(assets * totalSupply / totalAssets)`. Even when `totalSupply ≥ 1e18`, a large direct donation that inflates `totalAssets` can push the exchange rate so high that normal small deposits produce `sharesMinted == 0` due to integer truncation.

Latest OpenZeppelin `ERC4626` implementations support the virtual assets/shares pattern (adding a constant virtual offset to `totalAssets` and/or `totalSupply`) which effectively smooths the exchange rate and mitigates donation attacks by preventing extreme price skew. Despite this native mitigation, explicit non-zero-mint safeguards are still recommended to cover edge cases and future code changes.

Suggestion:

Add a `minShares` guard to guarantee a non-zero mint.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SUS1-5 Docs and Code Mismatch for Blacklist Administration

Severity: Minor

Status: Fixed

Code Location:

contracts/StakedUSDu.sol#105,119

Descriptions:

The public comments say “owner (DEFAULT_ADMIN_ROLE) and blacklist managers” can (un)blacklist; the implementation only allows `BLACKLIST_MANAGER_ROLE`.

Suggestion:

Allow the default admin to (un)blacklist.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

SUS1-6 Redundant Unvested Amount Calculation

Severity: Minor

Status: Acknowledged

Code Location:

contracts/StakedUSDu.sol#88,172

Descriptions:

`transferInRewards` re-computes `newVestingAmount = amount + getUnvestedAmount()` right after reverting if any unvested amount exists.

Suggestion:

Remove the `getUnvestedAmount()` in `newVestingAmount` calculation as it is always 0.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.