Unipay
Audit Report

Tue Jan 06 2026

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

ScaleBit

Unipay Audit Report

1 Executive Summary

1.1 Project Information

Description	The UnitasProxy contract functions as a centralized gateway for the Unitas protocol, enabling atomic interaction sequences for minting, staking, and redemption. It abstracts complex multi-step processes into single function calls, interacting with the core UnitasMintingV2 and StakedUSDu contracts.
Type	Yield Aggregator
Auditors	Fishmen,N0n3,Nebu1a
Timeline	Tue Jan 06 2026 - Tue Jan 06 2026
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/UnipayFI/boost-unitas-proxy/
Commits	fedf9b3a489d52f9361efd320e78cf9074714853

1.2 Files in Scope

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

ID	File	SHA-1 Hash
UPR1	contracts/UnitasProxy.sol	c5443fb97bf1e9738bc7aae36fa08e4c8b663af4

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	0	3
Centralization	1	0	1
Critical	0	0	0
Major	0	0	0
Medium	0	0	0
Minor	2	0	2
Informational	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 Unipay 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 3 issues of varying severity, listed below.

ID	Title	Severity	Status
UPR-3	Centralization Risks in `UnitasProxy`	Centralization	Acknowledged
UPR-4	Unrestricted Upgradeability Allows Immediate Malicious Logic Replacement	Minor	Acknowledged
UPR-5	Precision Loss in `flashWithdraw`	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Unipay Smart Contract :

1. Mint and Stake Flow

This operation allows users to mint `USDu` with collateral and immediately stake it for `sUSDu` in one atomic transaction.
Key Roles:

- **Caller (** `MINT_CALLER_ROLE` **)**: Typically a trusted relayer that submits the transaction.
- **Benefactor**: The user providing the collateral asset.
- **Beneficiary**: The recipient of the `sUSDu` shares.

Process Execution:

1. **Verification**: The Proxy validates that the order is strictly constructed for the Proxy's own address (`order.benefactor` and `order.beneficiary` must be `address(this)`).
2. **Collateral Collection**: The Proxy pulls the required `collateral_amount` from the User (Benefactor).
3. **Core Minting**: The Proxy invokes `minting.mint()` . `USDu` is minted to the Proxy.
4. **Auto-Staking**: The Proxy approves and deposits the Minted `USDu` into the Staking contract (`staked.deposit()`).
5. **Distribution**: `sUSDu` shares are minted directly to the User (Beneficiary).

2. Redeem and Withdraw Flow

This operation enables users to redeem their `USDu` positions back into the underlying collateral assets via the Proxy.
Key Roles:

- **Caller (** `REDEEM_CALLER_ROLE` **)**: Trusted relayer.
- **Benefactor**: The user burning `USDu` .

- **Beneficiary**: The recipient of the collateral asset.

Process Execution:

1. **Verification**: Similar to minting, the order must be scoped to the Proxy (`address(this)`).
2. **USDu Collection**: The Proxy pulls `order.usdu_amount` from the User (Benefactor).
3. **Core Redemption**: The Proxy invokes `minting.redeem()` , burning the `USDu` .
4. **Collateral Receipt**: The `UnitasMinting` contract releases collateral to the Proxy.
5. **Distribution**: The Proxy transfers the redeemed internal collateral to the User (Beneficiary).

3. Flash Withdraw Flow

This mechanism allows `sUSDu` holders to bypass the unbonding period by swapping `sUSDu` for `USDu` through a liquidity provider, subject to a penalty.

Process Execution:

1. **Exchange Rate Calculation**: The contract determines the amount of `USDu` underlying the provided `sUSDu` share amount based on the current exchange rate.
2. **Penalty Application**: A penalty fee (defined by `penaltyRate` , configured to 0.5% by default) is deducted from the `USDu` amount.
3. **Atomic Swap**:
    - **User -> MultiSig**: The User's `sUSDu` is transferred to the configured `multiSigWallet` (Liquidity Provider).
    - **MultiSig -> User**: The `multiSigWallet` transfers `USDu` (net of penalty) to the User.

4 Findings

UPR-3 Centralization Risks in `UnitasProxy`

Severity: Centralization

Status: Acknowledged

Code Location:

contracts/UnitasProxy.sol

Descriptions:

During the audit of the `UnitasProxy` contract, we identified certain design choices that rely heavily on the proper management of privileged roles. While these features facilitate user experience adjustments and operational flexibility, they also introduce centralization risks that depend on the robust security of the operational team's key management.

1. **Dependency on `MINT_CALLER_ROLE` for User Funds Security**: The `mintAndStake` function (and similarly `redeemAndWithdraw`) transfers funds from a `benefactor` user based on their ERC20 allowance to the Proxy contract. The function validates the transaction via a signature check against the `SIGNER_ROLE` (a protocol key) rather than a direct signature from the `benefactor`. Consequently, the security of user funds—specifically, ensuring funds are only moved with user intent—relies on the integrity of the off-chain system controlling the `MINT_CALLER_ROLE` and `SIGNER_ROLE`. If these keys were to be compromised, an attacker could potentially utilize the existing user allowances to move funds to an arbitrary `beneficiary` without a specific per-transaction signature from the user.
2. **Broad Capabilities of `rescueERC20`**: The `rescueERC20` function allows the `DEFAULT_ADMIN_ROLE` to transfer any amount of any ERC20 token held by the contract to any address. This function is typically intended for recovering tokens accidentally sent to the contract. However, its unrestricted nature means it technically functions as a mechanism that could withdraw any asset held by the contract, including user collateral that might be temporarily residing in the Proxy during a

transaction or other operational funds. The safety of these assets is therefore directly tied to the security of the `DEFAULT_ADMIN_ROLE`.

Suggestion:

It is recommended to consider the following enhancements to further decentralize trust and reduce reliance on key management:

1. For `mintAndStake` and similar flows, consider integrating EIP-712 signature verification. This would require the `benefactor` to sign the specific order parameters (including amounts and intended beneficiary) on-chain. This ensures that the Proxy can only execute transfers that have been explicitly authorized by the user for that specific context, removing the reliance on the Relayer's key for intent validation.
2. For `rescueERC20`, consider restricting the scope of recoverable tokens. For example, the contract could prevent the rescuing of the protocol's core tokens (like `USDu` or specific collateral assets) or limit the function's availability to only when the contract is paused. Alternatively, implementing a timelock or requiring a multi-signature governance process for this specific action would provide an additional layer of safety and transparency for protocol users.

Resolution:

Developer Response: These privileged addresses are either multisig wallets or executors within a TEE environment, so generally, there should be no issues.

UPR-4 Unrestricted Upgradeability Allows Immediate Malicious Logic Replacement

Severity: Minor

Status: Acknowledged

Code Location:

contracts/UnitasProxy.sol

Descriptions:

The `UnitasProxy` contract is designed as a Transparent Upgradeable Proxy (or similar UUPS/Beacon pattern relying on `OpenZeppelin` upgradeability libraries). This architecture separates the contract processing logic (Implementation) from the state (Proxy). The `Proxy Admin` role has the authority to update the address of the implementation contract. Currently, there is no time-lock mechanism enforcing a delay on upgrade operations. The entity controlling the `Proxy Admin` can presumably execute an `upgradeTo` or `upgradeToAndCall` transaction immediately.

If the `Proxy Admin` keys are compromised or if the entity acts maliciously, they can instantly replace the benign `UnitasProxy` implementation with a malicious contract. This malicious implementation could bypass all access controls, `rescueERC20` protections, or logic constraints, effectively granting the attacker full control over all assets held by the proxy and potentially corrupting the entire protocol state. This represents a single point of failure.

Suggestion:

It is recommended to transfer the ownership of the `Proxy Admin` (or the upgrade authority) to a `Timelock` contract.

A `Timelock` enforces a mandatory delay (e.g., 24 or 48 hours) between the proposal of a transaction (like an upgrade) and its execution. This delay provides transparent "Time to React" for the community and users. If a malicious or buggy upgrade is queued, users have

the opportunity to withdraw their funds or exit the protocol before the new logic becomes active.

Resolution:

Developer Response: We will subsequently transfer the ProxyAdmin to a multisig address.

UPR-5 Precision Loss in `flashWithdraw`

Severity: Minor

Status: Acknowledged

Code Location:

contracts/UnitasProxy.sol

Descriptions:

In the `flashWithdraw` function of `UnitasProxy.sol`, the penalty amount is calculated using standard integer arithmetic:

```
uint256 penaltyUsduAmount = (usduAmount * penaltyRate) / MAX_PENALTY_RATE;
```

Solidity's integer division involves truncation (rounding down). This means that any remainder from the division is discarded. Consequently, the calculated `penaltyUsduAmount` will effectively be rounded down in favor of the user (who pays less penalty).

For extremely small values of `usduAmount` (where `usduAmount * penaltyRate < MAX_PENALTY_RATE`), the calculated penalty would be zero.

Suggestion:

It is recommended to modify the penalty calculation logic to prioritize the protocol's interest by rounding up the penalty amount. This ensures that the protocol does not suffer from fee leakage due to precision truncation, especially for small withdrawal amounts.

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.