Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.10.23, the SlowMist security team received the Unitas team's security audit application for Unitas Contracts, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This is a stablecoin protocol where users can obtain USDu tokens by depositing collateral based on oracle-generated orders. The price of USDu and the order details are produced by the protocol backend and rigorously verified by the backend after being signed by users. Users can stake their USDu to receive sUSDu and accumulate rewards.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Redundant newVestingAmount	Design Logic Audit	Suggestion	Acknowledged

NO	Title	Category	Level	Status
	calculation			
N2	Order data validation relies on off-chain data integrity checks	Others	Information	Acknowledged
N3	Risks of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N4	Redundant order type validation in verifyRoute	Design Logic Audit	Suggestion	Acknowledged
N5	Inconsistent WETH asset processing logic	Design Logic Audit	Low	Acknowledged
N6	Potential risk of the `isActive` state being overwritten	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/UnipayFI/unitas-evm-contract

commit: f2ab1d38f1b7f0e14a5be200b811c0b461dafc80

Audit Scope:

```
./contracts
├── external
├── interfaces
├── lib
├── SingleAdminAccessControl.sol
├── StakedUSDu.sol
├── StakedUSDuV2.sol
├── UnitasMinting.sol
├── UnitasMintingV2.sol
├── USDu.sol
├── USDuSilo.sol
└── utils
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

SingleAdminAccessControl			
Function Name	Visibility	Mutability	Modifiers
transferAdmin	External	Can Modify State	onlyRole
acceptAdmin	External	Can Modify State	-
grantRole	Public	Can Modify State	onlyRole notAdmin
revokeRole	Public	Can Modify State	onlyRole notAdmin
renounceRole	Public	Can Modify State	notAdmin
owner	Public	-	-
_grantRole	Internal	Can Modify State	-

StakedUSDu			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC4626 ERC20Permit
transferInRewards	External	Can Modify State	nonReentrant onlyRole notZero
addToBlacklist	External	Can Modify State	onlyRole notOwner
removeFromBlacklist	External	Can Modify State	onlyRole notOwner
rescueTokens	External	Can Modify State	onlyRole
redistributeLockedAmount	External	Can Modify State	onlyRole
totalAssets	Public	-	-

StakedUSDu			
getUnvestedAmount	Public	-	-
decimals	Public	-	-
_checkMinShares	Internal	-	-
_deposit	Internal	Can Modify State	nonReentrant notZero notZero
_withdraw	Internal	Can Modify State	nonReentrant notZero notZero
_beforeTokenTransfer	Internal	Can Modify State	-
renounceRole	Public	Can Modify State	-

StakedUSDuV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	StakedUSDu
withdraw	Public	Can Modify State	ensureCooldownOff
redeem	Public	Can Modify State	ensureCooldownOff
unstake	External	Can Modify State	-
cooldownAssets	External	Can Modify State	ensureCooldownOn
cooldownShares	External	Can Modify State	ensureCooldownOn
setCooldownDuration	External	Can Modify State	onlyRole

UnitasMinting			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
mint	External	Can Modify State	nonReentrant onlyRole belowMaxMintPerBlock

UnitasMinting			
redeem	External	Can Modify State	nonReentrant onlyRole belowMaxRedeemPerBlock
setMaxMintPerBlock	External	Can Modify State	onlyRole
setMaxRedeemPerBlock	External	Can Modify State	onlyRole
disableMintRedeem	External	Can Modify State	onlyRole
setDelegatedSigner	External	Can Modify State	-
removeDelegatedSigner	External	Can Modify State	-
transferToCustody	External	Can Modify State	nonReentrant onlyRole
removeSupportedAsset	External	Can Modify State	onlyRole
isSupportedAsset	External	-	-
removeCustodianAddress	External	Can Modify State	onlyRole
removeMinterRole	External	Can Modify State	onlyRole
removeRedeemerRole	External	Can Modify State	onlyRole
addSupportedAsset	Public	Can Modify State	onlyRole
addCustodianAddress	Public	Can Modify State	onlyRole
getDomainSeparator	Public	-	-
hashOrder	Public	-	-
encodeOrder	Public	-	-
encodeRoute	Public	-	-
verifyOrder	Public	-	-
verifyRoute	Public	-	-

UnitasMinting			
verifyNonce	Public	-	-
_deduplicateOrder	Private	Can Modify State	-
_transferToBeneficiary	Internal	Can Modify State	-
_transferCollateral	Internal	Can Modify State	-
_setMaxMintPerBlock	Internal	Can Modify State	-
_setMaxRedeemPerBlock	Internal	Can Modify State	-
_computeDomainSeparator	Internal	-	-

UnitasMintingV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
mint	External	Can Modify State	nonReentrant onlyRole belowMaxMintPerBlock belowGlobalMaxMintPerBlock
mintWETH	External	Can Modify State	nonReentrant onlyRole belowMaxMintPerBlock belowGlobalMaxMintPerBlock
redeem	External	Can Modify State	nonReentrant onlyRole belowMaxRedeemPerBlock belowGlobalMaxRedeemPerBlock
setGlobalMaxMintPerBlock	External	Can Modify State	onlyRole
setGlobalMaxRedeemPerBlock	External	Can Modify State	onlyRole
disableMintRedeem	External	Can Modify State	onlyRole

UnitasMintingV2			
setDelegatedSigner	External	Can Modify State	-
confirmDelegatedSigner	External	Can Modify State	-
removeDelegatedSigner	External	Can Modify State	-
transferToCustody	External	Can Modify State	nonReentrant onlyRole
removeSupportedAsset	External	Can Modify State	onlyRole
isSupportedAsset	External	-	-
removeCustodianAddress	External	Can Modify State	onlyRole
removeMinterRole	External	Can Modify State	onlyRole
removeRedeemerRole	External	Can Modify State	onlyRole
removeCollateralManagerRole	External	Can Modify State	onlyRole
removeWhitelistedBenefactor	External	Can Modify State	onlyRole
addCustodianAddress	Public	Can Modify State	onlyRole
addWhitelistedBenefactor	Public	Can Modify State	onlyRole
setApprovedBeneficiary	Public	Can Modify State	-
getDomainSeparator	Public	-	-
hashOrder	Public	-	-
encodeOrder	Public	-	-
verifyOrder	Public	-	-
verifyRoute	Public	-	-
verifyNonce	Public	-	-

UnitasMintingV2			
verifyStablesLimit	Public	-	-
_deduplicateOrder	Private	Can Modify State	-
_transferToBeneficiary	Internal	Can Modify State	-
_transferCollateral	Internal	Can Modify State	-
_transferEthCollateral	Internal	Can Modify State	-
_setTokenConfig	Internal	Can Modify State	-
addSupportedAsset	External	Can Modify State	onlyRole
setMaxMintPerBlock	External	Can Modify State	onlyRole
_setMaxMintPerBlock	Internal	Can Modify State	-
setMaxRedeemPerBlock	External	Can Modify State	onlyRole
_setMaxRedeemPerBlock	Internal	Can Modify State	-
_computeDomainSeparator	Internal	-	-
setTokenType	External	Can Modify State	onlyRole
setStablesDeltaLimit	External	Can Modify State	onlyRole
_getDecimals	Internal	-	-
isCustodianAddress	Public	-	-
isWhitelistedBenefactor	Public	-	-
isApprovedBeneficiary	Public	-	-

USDu			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC20Permit
setMinter	External	Can Modify State	onlyOwner
mint	External	Can Modify State	-
renounceOwnership	Public	-	onlyOwner

USDuSilo			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
withdraw	External	Can Modify State	onlyStakingVault

4.3 Vulnerability Summary

[N1] [Suggestion] Redundant newVestingAmount calculation

Category: Design Logic Audit

Content

In the StakedUSDu contract's `transferInRewards` function, which is used to transfer rewards from the reward distributor to the staking contract, the function checks that `getUnvestedAmount()` must not be greater than 0 to ensure that no unvested rewards remain. However, when calculating `newVestingAmount`, it adds `getUnvestedAmount()` to `amount`. Since the previous check guarantees that `getUnvestedAmount()` is 0, this addition is redundant.

Code location: contracts/StakedUSDu.sol#L89-L90

```
  function transferInRewards(uint256 amount) external nonReentrant
onlyRole(REWARDER_ROLE) notZero(amount) {
    if (getUnvestedAmount() > 0) revert StillVesting();
    uint256 newVestingAmount = amount + getUnvestedAmount();
```

```
    ...
  }
```

Solution

It is recommended to assign `newVestingAmount` directly from the incoming `amount` without adding `getUnvestedAmount()`.

Status

Acknowledged

[N2] [Information] Order data validation relies on off-chain data integrity checks

Category: Others

Content

In the UnitasMinting / UnitasMintingV2 contracts, the `mint` and `redeem` functions are responsible for minting and redeeming stablecoins. The order data required for these operations is signed by the user, but the signed data originates from quotations generated by the project team's oracle service. After users sign, the backend performs strict validations to prevent data tampering.

This design pattern places critical data integrity verification in off-chain systems, introducing potential centralization and data manipulation risks. If the backend system is compromised or experiences a failure, it could result in incorrect minting operations, threatening both the protocol's security and user funds.

Code location:

contracts/UnitasMinting.sol#L161-L215

```
  function mint(Order calldata order, Route calldata route, Signature calldata
signature)
    external
    override
    nonReentrant
    onlyRole(MINTER_ROLE)
    belowMaxMintPerBlock(order.usdu_amount)
  {
    ...
  }

  function redeem(Order calldata order, Signature calldata signature)
    external
```

```
    override
    nonReentrant
    onlyRole(REDEEMER_ROLE)
    belowMaxRedeemPerBlock(order.usdu_amount)
  {
    ...
  }
```

contracts/UnitasMintingV2.sol#L234-L329

```
 function mint(Order calldata order, Route calldata route, Signature calldata
signature)
    external
    override
    nonReentrant
    onlyRole(MINTER_ROLE)
    belowMaxMintPerBlock(order.usdu_amount, order.collateral_asset)
    belowGlobalMaxMintPerBlock(order.usdu_amount)
  {
    ...
  }

  function mintWETH(Order calldata order, Route calldata route, Signature calldata
signature)
    external
    nonReentrant
    onlyRole(MINTER_ROLE)
    belowMaxMintPerBlock(order.usdu_amount, order.collateral_asset)
    belowGlobalMaxMintPerBlock(order.usdu_amount)
  {
    ...
  }

  function redeem(Order calldata order, Signature calldata signature)
    external
    override
    nonReentrant
    onlyRole(REDEEMER_ROLE)
    belowMaxRedeemPerBlock(order.usdu_amount, order.collateral_asset)
    belowGlobalMaxRedeemPerBlock(order.usdu_amount)
  {
    ...
  }
```

Solution

N/A

Status

Acknowledged

[N3] [Medium] Risks of excessive privilege

Category: Authority Control Vulnerability Audit

Content

In the UnitasMinting / UnitasMintingV2 contracts, the `transferToCustody` function is used to transfer assets to a custody wallet. Addresses with the `MINTER_ROLE` privilege can invoke this function to transfer any amount of any asset (including native ETH and ERC20 tokens) from the contract to any registered custody address. This design grants the `MINTER_ROLE` excessive privileges, creating a single-point-of-failure risk. If the private key associated with the `MINTER_ROLE` is compromised or maliciously abused, an attacker could transfer all assets from the contract to custody addresses, resulting in user fund losses. Similarly, the `mint` and `redeem` operations depend on both the `MINTER_ROLE` and `REDEEMER_ROLE`, which also carry risks of excessive privilege.

The project team mitigates such risks by imposing per-block limits on the mintable/redeemable amounts and by introducing the `GATEKEEPER_ROLE` for emergency intervention.

Solution

The `DEFAULT_ADMIN_ROLE` is responsible for managing the privileged roles mentioned above. To prevent single-point-of-failure risks, this role should be managed via a multisignature wallet. In the long term, combining multisig management with a timelock mechanism for delayed transaction execution would effectively mitigate the risks associated with excessive privileges.

Status

Acknowledged

[N4] [Suggestion] Redundant order type validation in verifyRoute

Category: Design Logic Audit

Content

In the UnitasMinting contract, the `verifyRoute` function is used to validate the legitimacy of a routing object. However, it's worth noting that the `redeem` function does not invoke `verifyRoute` —only the `mint` function does.

Despite this, `verifyRoute` still includes a check for `OrderType.REDEEM` and returns `true` directly. This check is therefore entirely redundant and adds unnecessary code complexity.

Code location: contracts/UnitasMinting.sol#L352

```
 function verifyRoute(Route calldata route, OrderType orderType) public view override
returns (bool) {
    // routes only used to mint
    if (orderType == OrderType.REDEEM) {
      return true;
    }
    ...
  }
```

Solution

It is recommended to remove the redundant `OrderType.REDEEM` check from the `verifyRoute` function.

Status

Acknowledged

[N5] [Low] Inconsistent WETH asset processing logic

Category: Design Logic Audit

Content

In the UnitasMintingV2 contract, the `_transferCollateral` function is responsible for transferring collateral to an array of custody addresses. If the user deposits WETH, the transfer should be handled via the `mintWETH` function, which performs the transfer through `_transferEthCollateral`. However, the current `_transferCollateral` implementation only checks whether the asset is not equal to `NATIVE_TOKEN` and fails to verify whether the asset corresponds to the WETH address. As a result, if WETH is processed directly through `_transferCollateral`, it bypasses the conversion logic from WETH to ETH, causing the custody address to receive WETH tokens instead of native ETH. This may lead to inconsistencies in fund management and accounting processes.

Code location: contracts/UnitasMintingV2.sol#L632

```
function _transferCollateral(
    ...
 ) internal {
    // cannot mint using unsupported asset or native ETH even if it is supported for
```

```
redemptions
    if (!tokenConfig[asset].isActive || asset == NATIVE_TOKEN) revert
UnsupportedAsset();
    ...
  }
```

Solution

Add a WETH address check within the `_transferCollateral` function to ensure that WETH assets can only be processed through the `mintWETH` and `_transferEthCollateral` functions.

Status

Acknowledged

[N6] [Suggestion] Potential risk of the `isActive` state being overwritten

Category: Design Logic Audit

Content

In the UnitasMintingV2 contract, the `_setTokenConfig` function is used to configure token asset parameters. The current implementation first sets the `isActive` field of the incoming `_tokenConfig` parameter to `true`, and then assigns the entire `_tokenConfig` struct to `tokenConfig[asset]`. If the `isActive` field in the passed `_tokenConfig` is `false`, this assignment overwrites the previously set `isActive` value, causing the token configuration to be marked as inactive.

Code location: contracts/UnitasMintingV2.sol#L684-L685

```
function _setTokenConfig(address asset, TokenConfig memory _tokenConfig) internal {
    if (_tokenConfig.maxMintPerBlock == 0 || _tokenConfig.maxRedeemPerBlock == 0) {
      revert InvalidAmount();
    }
    _tokenConfig.isActive = true;
    tokenConfig[asset] = _tokenConfig;
  }
```

Solution

Reorder the operations by first assigning `_tokenConfig` to `tokenConfig[asset]`, and then explicitly setting `isActive` to `true`.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002510290002	SlowMist Security Team	2025.10.23 - 2025.10.29	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we found 1 medium risk, 1 low risk, 3 suggestions, and 1 information. All the findings were acknowledged. The code was not deployed to the mainnet. Since stablecoin minting and redemption operations rely entirely on the protocol’s backend, the protocol remains exposed to the risk of excessive privilege. Therefore, the final assessment is classified as medium risk.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist