Unipay
Audit Report

Mon Mar 10 2025

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

ScaleBit

Unipay Audit Report

1 Executive Summary

1.1 Project Information

Description	This project implements a stablecoin system with the following key components: - USDU: The main stablecoin token - SUSDU: Staking token for USDU - Vault: Manages collateral, minting, and redemption
Type	Stablecoin
Auditors	ScaleBit
Timeline	Fri Feb 21 2025 - Mon Mar 10 2025
Languages	Rust
Platform	Solana
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/UnipayFI/stablecoin/
Commits	e6300e337ce17bc428e99dd2e50761cdcff6faa5 0d21f3af3704ad180f39a2727c6429f25512f5a6 055a4bddf66748e022651037402092323 11d6e8c

1.2 Files in Scope

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

ID	File	SHA-1 Hash
TOK	programs/vault/src/utils/token.rs	6975b91a55ebef1db436caa8ddb32fb92218e0f9
ROU	programs/vault/src/math/rounding.rs	5a9ef46f6b72334f8bc6b8dc94ed4cfea4a9ca0b
PVSMER	programs/vault/src/math/error.rs	cfde7389cfa441153f39f369b533fd90a64cafc1
COO	programs/vault/src/state/cooldown.rs	a7b77d0795997bd5980181db3790213e9a0cff10
IVA	programs/vault/src/instructions/admin/init_vault.rs	12a3206de7c76d7745c419ef1c3b35a2db33bddc
PVSISMR	programs/vault/src/instructions/susdu/mod.rs	5cd66389f3745786df43b7516309dcf15f4bade7
PVSIMR	programs/vault/src/instructions/mod.rs	0a5d8d8b0fb0b6b22dd17344883c2e3b58378a3d
PVSIUMR	programs/vault/src/instructions/usdu/mod.rs	3c0f445cef22e92d45c3926cbc86235df4a01705
PSSSCR	programs/susdu/src/state/config.rs	0410dfb32778af39103ddc72061eec850b3309d4
PSSSMR	programs/susdu/src/state/mod.rs	ee57b90e078514c10360966a2336a991fb3b16c3
ICO	programs/susdu/src/instructions/admin/init_config.rs	b7c033d534aec7af64c50c11fd9627809e33d9c0

PSSIAMR	programs/susdu/src/instructions/admin/mod.rs	67eb838e2038ae80b0d5fdfcbc20ddc8ae75269c
PSSIMR	programs/susdu/src/instructions/mod.rs	bd07f7b08a5a7515b15d8759434c8c0b405edd80
PUSCR	programs/usdu/src/constants.rs	a2b0d2358ddcee936bf3cdac0e914986b392ba9a
PUSSCR	programs/usdu/src/state/config.rs	2a4aede9917a7f0b648a3d383772da05cddc8aad
PUSIAICR	programs/usdu/src/instructions/admin/init_config.rs	765fb822a77a374f743a1ade0cdd7764bd0242b5
PUSIMR	programs/usdu/src/instructions/mod.rs	66068ae07885dc8f2a80b429a7b1a3a5f1051827
PGSCR	programs/guardian/src/constants.rs	f98041b722aa43faa08c3fffa11349f1283be8c8
PGSER	programs/guardian/src/events.rs	0a6183801b967e27ae17c06421437d95301aed66
PGSSMR	programs/guardian/src/state/mod.rs	a8481321ccf8669161e15bc206368e27b1c0b04f
PGSIAMR	programs/guardian/src/instructions/admin/mod.rs	6b0cecba3368dca22b5865f0da197f705f10069e
IAR	programs/guardian/src/instructions/admin/init_access_registry.rs	e7abd660e5f230f87942b4414e982daf8d213618
PGSIMR	programs/guardian/src/instructions/mod.rs	ec4223a4f15ec64b4fc771efabdb87a8431f77bf

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	19	16	3
Informational	3	3	0
Minor	5	5	0
Medium	4	1	3
Major	6	6	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

MoveBit 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 by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

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) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) 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 Unipay 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 19 issues of varying severity, listed below.

ID	Title	Severity	Status
ABL-1	`blacklist_state` Init Constraint Blocks Modifications	Major	Fixed
ACO-1	Redundant Event Emission on No Change	Minor	Fixed
ARO-1	Duplicate Check For `access_registry.admin`	Informational	Fixed
CON-1	Precision Loss and Potential Exploitation of Share Calculation	Medium	Acknowledged
DCM-1	Centralization Risk	Medium	Acknowledged
DUR-1	Potential Ambiguity in `get_unvested_amount` Comparison	Minor	Fixed
EME-1	Missing Update to `vault_config` in `process_emergency_xxxx`	Major	Fixed
IAR-1	Lack of Access Control on Initialization Function	Medium	Acknowledged
MUS-1	Possible Overflow Risk in	Medium	Fixed

	`usdu_config.total_supply`		
RLO-1	Lack of Restrictions on Locked Shares	Major	Fixed
SUM-1	Missing Constraints on `blacklist_state`	Major	Fixed
SUM-2	Duplicate Checks on Some Accounts	Informational	Fixed
TAD-1	Redundant Event Emission on No Change	Minor	Fixed
THO-1	Should Throw an `IsNotTransferring` Error	Minor	Fixed
USU-1	Missing Account Reloading After CPI	Major	Fixed
USU-2	Multiple or Unnecessary Checks on `cooldown`	Minor	Fixed
USU-3	Duplicate Check on `caller_susdu_token_account.amount`	Informational	Fixed
UTI-1	Invalid Check Logic in `has_privileged_role`	Major	Fixed
UTI-2	Possible redundant checks	Discussion	Fixed

3 Participant Process

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

0. Role Assignment

```
// Assign roles to vault_config
usduMinter = await AssignRole(guardianProgram, accessRegistry, admin, vaultConfig,
"usdu_minter");
susduMinter = await AssignRole(guardianProgram, accessRegistry, admin, vaultConfig,
"susdu_minter");
usduRedeemer = await AssignRole(guardianProgram, accessRegistry, admin,
vaultConfig, "usdu_redeemer");
susduRedeemer = await AssignRole(guardianProgram, accessRegistry, admin,
vaultConfig, "susdu_redeemer");
susduRedistributor = await AssignRole(guardianProgram, accessRegistry, admin,
vaultConfig, "susdu_redistributor");

// Assign roles to admin
collateralDepositor = await AssignRole(guardianProgram, accessRegistry, admin,
admin.publicKey, "collateral_depositor");
collateralWithdrawer = await AssignRole(guardianProgram, accessRegistry, admin,
admin.publicKey, "collateral_withdrawer");
grandMaster = await AssignRole(guardianProgram, accessRegistry, admin,
admin.publicKey, "grand_master");
```

1. Initial Setup

```
// Initialize core components
await InitGuardianAccessRegistry(guardianProgram, accessRegistry, admin);
await InitAndCreateUSDU(usduProgram, usduMintToken, accessRegistry, usduConfig,
admin);
await InitAndCreateSusdu(susduProgram, susduMintToken, accessRegistry,
susduConfig, admin);
await InitVaultConfig(vaultProgram, vaultConfig, accessRegistry, usduMintToken,
susduMintToken, admin, 0);
await InitVaultState(
  vaultProgram,
  vaultConfig,
  vaultState,
  vaultUsduTokenAccount,
  vaultSusduTokenAccount,
  vaultStakePoolUsduTokenAccount,
```

```
    vaultSlioUsduTokenAccount,
    usduMintToken,
    susduMintToken,
    admin
);
```

2. Main Operations

2.1 Deposit Collateral and Mint USDU

```
// Mint collateral tokens
const mintIx = createMintToInstruction(
    mintToken.publicKey,
    benefactorCollateralTokenAccount.address,
    admin.publicKey,
    100_000_000_000,
    [admin],
    TOKEN_2022_PROGRAM_ID
);

// Approve collateral tokens
const approveIx = createApproveInstruction(
    benefactorCollateralTokenAccount.address,
    vaultConfig,
    benefactor.publicKey,
    100_000_000_000,
    [benefactor],
    TOKEN_2022_PROGRAM_ID
);

// Deposit and mint USDU
await DepositCollateralAndMintUsdu(
    vaultProgram,
    usduProgram,
    admin,
    vaultConfig,
    usduConfig,
    accessRegistry,
    usduMinter,
    collateralDepositor,
    mintToken.publicKey,
    usduMintToken,
    benefactor,
    beneficiary,
```

```
    fund,
    1200_000_000,
    1000_000_000,
    benefactorCollateralTokenAccount.address,
    beneficiaryUsduTokenAccount.address,
    fundCollateralTokenAccount.address,
    vaultCollateralTokenAccount,
);
```

2.2 Redeem USDU and Withdraw Collateral

```
// Approve collateral to the vault
const approveCollateralIx = createApproveInstruction(
    fundCollateralTokenAccount.address,
    vaultConfig,
    fund.publicKey,
    100_000_000_000,
    [fund],
    TOKEN_2022_PROGRAM_ID
);

// Approve USDU to the vault
const approveUsduIx = createApproveInstruction(
    beneficiaryUsduTokenAccount.address,
    vaultConfig,
    beneficiary.publicKey,
    60_000_000_000,
    [beneficiary],
    TOKEN_2022_PROGRAM_ID
);

// Redeem and withdraw
await RedeemUsduAndWithdrawCollateral(
    vaultProgram,
    usduProgram,
    admin,
    vaultConfig,
    vaultState,
    usduConfig,
    accessRegistry,
    usduRedeemer,
    collateralWithdrawer,
    mintToken.publicKey,
```

```
    usduMintToken,
    benefactor,
    beneficiary,
    fund,
    20_000_000,
    10_000_000,
    beneficiaryUsduTokenAccount.address,
    fundCollateralTokenAccount.address,
    vaultUsduTokenAccount,
    benefactorCollateralTokenAccount.address,
);
```

2.3 Stake USDU and Mint SUSDU

```
const blacklistState = PublicKey.findProgramAddressSync(
    [Buffer.from(vaultBlacklistSeed), caller.publicKey.toBuffer()],
    vaultProgram.programId
)[0];

await StakeUsduMintSusdu(
    vaultProgram,
    susduProgram,
    caller,
    susduReceiver,
    susduReceiverSusduTokenAccount.address,
    beneficiaryUsduTokenAccount.address,
    accessRegistry,
    vaultStakePoolUsduTokenAccount,
    susduMinter,
    usduMintToken,
    susduMintToken,
    vaultState,
    vaultConfig,
    susduConfig,
    100_000_000,
    blacklistState,
);
```

2.4 Unstake SUSDU

```
const [cooldown] = PublicKey.findProgramAddressSync(
    [
        Buffer.from(vaultCooldownSeed),
```

```
      Buffer.from(usduMintToken.toBuffer()),
      Buffer.from(receiver.publicKey.toBuffer()),
    ],
    vaultProgram.programId
);

await UnstakeSusdu(
    vaultProgram,
    susduProgram,
    caller,
    callerSusduTokenAccount,
    receiver.publicKey,
    receiverUsduTokenAccount,
    susduConfig,
    vaultConfig,
    vaultState,
    vaultSusduTokenAccount,
    cooldown,
    accessRegistry,
    susduRedeemer,
    susduMintToken,
    usduMintToken,
    vaultStakePoolUsduTokenAccount,
    vaultSlioUsduTokenAccount,
    30_000_000,
    blacklistState,
);
```

2.5 Withdraw USDU (After Cooldown)

```
await WithdrawUsdu(
    vaultProgram,
    receiver,
    vaultConfig,
    vaultState,
    receiverUsduTokenAccount,
    vaultSlioUsduTokenAccount,
    cooldown,
    usduMintToken,
    blacklistState,
);
```

2.6 Distribute USDU Reward

```
const distributeRewarder = await AssignRole(
    guardianProgram,
    accessRegistry,
    admin,
    beneficiary.publicKey,
    "distribute_rewarder"
);

await DistributeUsduReward(
    vaultProgram,
    caller,
    vaultConfig,
    vaultState,
    callerUsduTokenAccount,
    vaultStakePoolUsduTokenAccount,
    accessRegistry,
    distributeRewarder,
    usduMintToken,
    susduMintToken,
    susduConfig,
    100_100_000,
);
```

2.7 Blacklist Management

```
// Adjust blacklist
await AdjustBlacklist(
    vaultProgram,
    admin,
    vaultConfig,
    accessRegistry,
    grandMaster,
    true,  // add to blacklist
    false, // is permanent
    user,
);

// Redistribute locked SUSDU
await RedistributeLockedSusdu(
    vaultProgram,
    susduProgram,
    vaultConfig,
```

```
    admin,
    accessRegistry,
    grandMaster,
    susduRedistributor,
    susduConfig,
    susduMintToken,
    lockedSusduTokenAccount,
    newSusduReceiverTokenAccount.address,
    blacklistState,
    newSusduReceiver.publicKey,
);
```

4 Findings

ABL-1 `blacklist_state` Init Constraint Blocks Modifications

Severity: Major

Status: Fixed

Code Location:

programs/vault/src/instructions/admin/adjust_blacklist.rs#64-66

Descriptions:

In the `process_adjust_blacklist` function, there is a constraint on `blacklist_state.is_initialized == false`, but this makes it impossible to cancel or modify the blacklist for a user.

Suggestion:

Suggest removing the `blacklist_state.is_initialized` constraint from `process_adjust_blacklist` function while preserving access control.

Resolution:

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

ACO-1 Redundant Event Emission on No Change

Severity: Minor

Status: Fixed

Code Location:

programs/vault/src/instructions/admin/adjust_cooldown.rs#16

Descriptions:

Currently, if the `cooldown_duration` value is the same as the previous value, the event `CooldownAdjusted` will still be emitted. This could mislead users into thinking that the cooldown duration has been updated when it has not.

Suggestion:

Before emitting the event, check if the new `cooldown_duration` is different from the existing one. Only emit the event if the value has actually changed. This avoids unnecessary event emissions and prevents misleading information.

Resolution:

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

ARO-1 Duplicate Check For `access_registry.admin`

Severity: Informational

Status: Fixed

Code Location:

programs/guardian/src/instructions/admin/assign_role.rs#16-23;

programs/guardian/src/instructions/admin/revoke_role.rs#12-19

Descriptions:

Duplicate check for `access_registry.admin` in **AssignRole** and **RevokeRole**.

```
  #[account(
    mut,
    has_one = admin,
    seeds = [ACCESS_REGISTRY_SEED],
    bump = access_registry.bump,
    constraint = access_registry.is_initialized @
GuardianError::AccessRegistryNotInitialized,
    constraint = access_registry.admin == admin.key() @
GuardianError::MustBeAccessRegistryAdmin //@audit double-check
  )]
```

Suggestion:

Suggest removing duplicate checks.

Resolution:

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

CON-1 Precision Loss and Potential Exploitation of Share Calculation

Severity: Medium

Status: Acknowledged

Code Location:

programs/usdu/src/state/config.rs#96

Descriptions:

The `convert_to_shares` function uses a formula derived from the EIP-4626 standard for calculating shares. However, since the share calculation involves rounding down, users may always receive fewer shares than they are entitled to, leading to a loss of precision. The discrepancy between the actual shares and the shares they should receive is determined by the exchange rate between the shares and the underlying tokens. An attacker can exploit this by sending a large amount of assets to the contract via the `distribute_usdu_reward()` function, which could increase the share-to-asset exchange rate and magnify the precision loss. This could lead to user asset loss, especially when the share amount becomes so small that users no longer receive any shares.

Suggestion:

To mitigate this issue,we highly recommend:

1. Add a sufficient initial deposit of assets to the contract, which can increase the cost of an attack. This is similar to mechanisms seen in some staking contracts, where a minimum asset threshold is required to participate.
2. Apply a precision offset between the shares and assets to limit the loss, ensuring that the losses due to rounding are minimized.

Resolution:

1. `stake_usdu_mint_susdu.rs` , already add `check_inital_deposit` function, and `INITIAL_DEPOSIT_AMOUNT` is 1000

```
// 3. check usdu amount and initial deposit
require!(usdu_amount > 0, VaultError::InvalidStakeUsduAmount);
```

```
let vault_config = &mut ctx.accounts.vault_config;
vault_config.check_initial_deposit(usdu_amount)?;
```

2. `distribute_usdu_reward` function need `RewardDistributor` role or admin
3. according to Ethena `convert_to_share` and `convert_to_asset` function

DCM-1 Centralization Risk

Severity: Medium

Status: Acknowledged

Code Location:

programs/vault/src/instructions/usdu/deposit_collateral_mint_usdu.rs;

programs/vault/src/instructions/usdu/redeem_usdu_withdraw_collateral.rs;

programs/vault/src/instructions/admin/redistribute_locked.rs

Descriptions:

Centralization risk was identified in the smart contract:

- The exchange rate between `collateral_token` and `usdu_token` is determined by `CollateralDepositor`
- `GrandMaster` can lock any account and withdraw locked shares. Locked accounts can no longer be used for stake, exchanges and other activities.
- The behavior of `access_registry.admin` is not restricted in any way, so malicious administrators may cause irreparable damage to assets.

Suggestion:

It is recommended that measures be taken to reduce the risk of centralization, such as a multi-signature mechanism.

Resolution:

We will use multi-signature account after deployment, transfer admin to multi-sig account

DUR-1 Potential Ambiguity in `get_unvested_amount` Comparison

Severity: Minor

Status: Fixed

Code Location:

programs/vault/src/instructions/admin/distribute_usdu_reward.rs#77

Descriptions:

In the condition `require!(vault_config.get_unvested_amount() <= 0, VaultError::StillVesting);`, the use of `<=` could lead to ambiguous interpretations. Since `get_unvested_amount()` returns a `u64` value, this check implies that it can be greater than or equal to zero, which is always true for unsigned integers. This may not clearly express the intention that the unvested amount should be exactly zero.

Suggestion:

Replace the `<=` with `==` to explicitly check if the unvested amount is exactly zero. This will improve clarity and ensure that the vesting process is correctly handled only when there are no unvested amounts left.

```
require!(vault_config.get_unvested_amount() == 0, VaultError::StillVesting);
```

Resolution:

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

EME-1 Missing Update to `vault_config` in `process_emergency_xxxx`

Severity: Major

Status: Fixed

Code Location:

programs/vault/src/instructions/admin/emergency.rs

Descriptions:

Missing updates to `vault_config.total_usdu_supply` fields in `process_emergency_xxxx` series of functions after removing usdu, susdu from vault.

Suggestion:

Suggest updating `vault_config` after refunding

Resolution:

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

IAR-1 Lack of Access Control on Initialization Function

Severity: Medium

Status: Acknowledged

Code Location:

programs/guardian/src/instructions/admin/init_access_registry.rs#24

Descriptions:

The `process_init_access_registry` function can be called by any user, potentially allowing an attacker to initialize the `access_registry`. This could be a security risk, especially if this function is called after contract deployment but before the intended initialization, allowing an attacker to control the access registry or perform unintended actions.

```
#[derive(Accounts)]
pub struct InitAccessRegistry<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

    #[account(
        init_if_needed,
        payer = admin,
        space = AccessRegistry::SIZE,
        seeds = [ACCESS_REGISTRY_SEED],
        bump
    )]
    pub access_registry: Account<'info, AccessRegistry>,
    pub system_program: Program<'info, System>,
}
```

The function checks if the registry is already initialized but does not restrict who can call the function. Without proper access control, the function could be invoked by an attacker, potentially leading to unauthorized access or misuse of the contract.

Suggestion:

This problem only occurs when deployment and initialization are not executed in the same transaction. If your script deployment and initialization are executed at the same time, this problem will not occur.

Otherwise,it is recommended to implement access control to ensure that only authorized users (such as the contract deployer or a specific admin address) can call this function. You can add a check to ensure that only a specific address, such as the admin, can call the initialization function.

For example:

```
require!(ctx.accounts.admin.key() == AUTHORIZED_ADDRESS,
GuardianError::Unauthorized);
```

Where `AUTHORIZED_ADDRESS` is the address allowed to initialize the registry.

Resolution:

1. add `admin` and `is_initialize` field in `AccessRegistryInitialized` , to check acutal admin address and initialization status.
2. will do init access_registry right after deployment.

MUS-1 Possible Overflow Risk in `usdu_config.total_supply`

Severity: Medium

Status: Fixed

Code Location:

programs/usdu/src/instructions/mint_usdu.rs#71

Descriptions:

In the `process_mint_usdu` function, `usdu_config.total_supply` may overflow, and while this will throw an error on mint_to (due to its synchronization with `usdu_token.total_supply`), it makes it impossible to continue minting coins.

Suggestion:

It is recommended to use whitelisting for `collateral_token` and to limit the issuance of `usdu_token`.

Resolution:

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

RLO-1 Lack of Restrictions on Locked Shares

Severity: Major

Status: Fixed

Code Location:

programs/vault/src/instructions/admin/redistribute_locked.rs;

programs/vault/src/instructions/admin/adjust_blacklist.rs

Descriptions:

We note that the `process_redistribute_locked` function is used to reclaim the shares of a blacklisted `locked_susdu_token_account`, but locked accounts can transfer shares via the transfer method of `spl_token_2022 itself`, which causes the blacklisting mechanism to This will cause the blacklisting mechanism to be ineffective.

Suggestion:

It is recommended to improve the mechanism of blacklisting, e.g. to create a new token account for the corresponding user while adding a blacklist, and to transfer the shares from the locked account to the new account. This way, the banned user can no longer transfer the locked shares.

Resolution:

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

SUM-1 Missing Constraints on `blacklist_state`

Severity: Major

Status: Fixed

Code Location:

programs/vault/src/instructions/susdu/stake_usdu_mint_susdu.rs#93,116-119;
programs/vault/src/instructions/susdu/unstake_susdu.rs#145-148;
programs/vault/src/instructions/susdu/withdraw_usdu.rs#66-68;
programs/vault/src/instructions/admin/redistribute_locked.rs#92-95

Descriptions:

Functions involving blacklist detection, such as `process_stake_usdu_mint_susdu`, lack a check on the `blacklist_state`, which means that a user can bypass blacklist detection by providing a fake `blacklist_state`.

Suggestion:

Suggest adding constraints on `blacklist_state`

Resolution:

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

SUM-2 Duplicate Checks on Some Accounts

Severity: Informational

Status: Fixed

Code Location:

programs/vault/src/instructions/susdu/stake_usdu_mint_susdu.rs#76,121-124;

programs/vault/src/instructions/usdu/redeem_usdu_withdraw_collateral.rs#24,123-126

Descriptions:

Duplicate check for `vault_state.vault_stake_pool_usdu_token_account` in `process_stake_usdu_mint_susdu` function

Duplicate check for `vault_state.vault_usdu_token_account` in `process_redeem_usdu_withdraw_collateral` function

Suggestion:

Suggest removing duplicate checks.

Resolution:

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

TAD-1 Redundant Event Emission on No Change

Severity: Minor

Status: Fixed

Code Location:

programs/blacklist-hook/src/instructions/transfer_admin.rs#43-77;

programs/susdu/src/instructions/admin/update_transfer_hook.rs#27-58

Descriptions:

For updates to `susdu_config.blacklist_hook_program_id` and `blacklist_hook_config.pending_admin`, if the new value matches the existing value, neither an update nor an event emission should be triggered.

Suggestion:

Only emit the event if the value has actually changed.

Resolution:

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

THO-1 Should Throw an `IsNotTransferring` Error

Severity: Minor

Status: Fixed

Code Location:

programs/blacklist-hook/src/instructions/transfer_hook.rs#57-59

Descriptions:

In the `process_transfer_hook` function, if `account_with_extensions.transferring` is false, it proves that the call is not in the transfer process and should throw an `IsNotTransferrin` g error.

Suggestion:

Suggests throwing an `IsNotTransferring` error.

Resolution:

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

USU-1 Missing Account Reloading After CPI

Severity: Major

Status: Fixed

Code Location:

programs/vault/src/instructions/susdu/unstake_susdu.rs#229-245,264;
programs/vault/src/instructions/susdu/stake_usdu_mint_susdu.rs#163-184

Descriptions:

The `process_unstake_susdu` function does not reload `susdu_config` after updating `susdu_config.total_supply` via a CPI call to `redeem_susdu`, which results in an outdated total_supply being passed to check_min_shares. The same problem exists in the `process_stake_usdu_mint_susdu` function

Suggestion:

Suggest calling Anchor's reload method on the account. This will refresh the struct's fields with the current underlying data.

Resolution:

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

USU-2 Multiple or Unnecessary Checks on `cooldown`

Severity: Minor

Status: Fixed

Code Location:

programs/vault/src/instructions/susdu/unstake_susdu.rs#72-80,198-226

Descriptions:

Duplicate check for `cooldown.owner` in `process_unstake_susdu` function. Additionally, the constraint `cooldown.is_initialized == false` contradicts the following code logic:

```
// 10. check cooldown is initialized
if !ctx.accounts.cooldown.is_initialized {
    let cooldown = Cooldown {
        is_initialized: true,
        cooldown_end: Clock::get().unwrap().unix_timestamp as u64
            + vault_config.cooldown_duration,
        underlying_token_account: ctx
            .accounts
            .receiver_usdu_token_account
            .to_account_info()
            .key(),
        underlying_token_mint: ctx.accounts.usdu_token.to_account_info().key(),
        underlying_token_amount: usdu_amount,
        owner: ctx.accounts.caller.key(),
        bump: ctx.bumps.cooldown,
    };
    ctx.accounts.cooldown.set_inner(cooldown);
} else {
    require!(
        ctx.accounts.cooldown.owner == ctx.accounts.caller.key(),
        VaultError::InvalidCooldownOwner
    );
    let cooldown = &mut ctx.accounts.cooldown;

    // Consistent with Ethena protocol, reset cooldown period for each unstake
    cooldown.cooldown_end =
        Clock::get().unwrap().unix_timestamp as u64 + vault_config.cooldown_duration;
```

```
    cooldown.underlying_token_amount += usdu_amount;
};
```

Suggestion:

Suggest removing multiple checks and confirming checks that are unnecessary.

Resolution:

Fixed by removing `constraint = false || cooldown.owner == caller.key() @ VaultError::InvalidCooldownOwner`

USU-3 Duplicate Check on `caller_susdu_token_account.amount`

Severity: Informational

Status: Fixed

Code Location:

programs/vault/src/instructions/susdu/unstake_susdu.rs#169-172;

programs/vault/src/instructions/susdu/unstake_susdu.rs#184-187

Descriptions:

Duplicate Check on `caller_susdu_token_account.amount` in `process_unstake_susdu` function.

Suggestion:

Suggest removing duplicate check.

Resolution:

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

UTI-1 Invalid Check Logic in `has_privileged_role`

Severity: Major

Status: Fixed

Code Location:

programs/guardian/src/utils.rs#61-97

Descriptions:

A default return value from `Pubkey::find_program_address` does not imply that the user has the corresponding role.

```
for role in privileged_roles.iter() {
    let (role_address, _) = Pubkey::find_program_address(
        &[
            ACCESS_ROLE_SEED,
            access_registry.key().as_ref(),
            user.as_ref(),
            role.to_seed().as_slice(),
        ],
        &crate::ID,
    );

    if role_address != Pubkey::default() {
        return Ok(true);
    }
```

Suggestion:

Suggest further verifying the role_address account data.

Resolution:

Seem useless funciton, already remove it.

UTI-2 Possible redundant checks

Severity: Discussion

Status: Fixed

Code Location:

programs/guardian/src/utils.rs#41-43

Descriptions:

In the has_role function, we find that some of the checks in the following code may be redundant. We would like to know what the checks on `matched_role.owner` and `matched_role.role` are intended to prevent.

```
  if access_role.key() == role_address || (authority.key() == matched_role.owner &&
matched_role.role == role) {
      return Ok(true);
  }
```

Suggestion:

Suggest removing redundant checks, if confirmed.

Resolution:

already remove `(authority.key() == matched_role.owner && matched_role.role == role)` condition

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.