DFORCE SUSX SECURITY AUDIT REPORT

Aug 12, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION 3

1.1 Disclaimer 3

1.2 Security Assessment Methodology 3

1.3 Project Overview 7

1.4 Project Dashboard 8

1.5 Summary of findings 11

1.6 Conclusion 13

2.FINDINGS REPORT 15

2.1 Critical 15

2.2 High 15

H-1 MintCap Check Missing in Cross-Chain Transfer 15

2.3 Medium 17

M-1 Inaccurate `maxWithdraw` and `maxRedeem` Function Results 17

M-2 Missing Function for Future Config Edits 18

M-3 Potential Arbitrage Due to Rate Discrepancies in Cross-Chain Transfers 19

M-4 Centralization Risks 20

M-5 Potential Deposit Blockage 21

M-6 Inability to perform a cross-chain transfer due to mintCap 22

M-7 `deposit()` doesn't check future rewards from configured epochs 23

2.4 Low 24

L-1 `updateEpochId` gas optimization 24

L-2 Config range checks 25

L-3 The epoch configuration may overlap by 1 second 26

L-4 Unchecked in `for` loop is default behaviour since 0.8.22 27

L-5 Optimization Issues in Function `_rpow` 28

L-6 Inefficient Initializer Usage 29

L-7 Double `Ownable2Step` Initialization 30

L-8 Missing Zero-Address Checks 31

L-9 Incorrect Event Parameter in Function `_withdraw` 32

L-10 Incorrect Condition in Function `currentAPY` 33

L-11 Gas optimisation in `mul` and `rmul` 34

L-12 Blocking users' deposits by front-running 35

3. ABOUT MIXBYTES 36

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

DForce sUSX is a staking contract for the USX token. It provides admin-defined staking rewards for stakers. Rewards are minted as USX tokens.

1.4 Project Dashboard

Project Summary

Title	Description
Client	dForce
Project name	sUSX
Timeline	13.06.2024 - 27.06.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
13.06.2024	2ff5d7a8f9a509006557bf5de72fabf40d1138a5	Initial commit for the audit
24.06.2024	d442b32e54f9a6c63896a2a0a5fc8f4f0f0fb001	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
src/library/ProxyAdmin2Step.sol	ProxyAdmin2Step.sol
src/library/RateMath.sol	RateMath.sol
src/sUSX.sol	sUSX.sol
src/USR.sol	USR.sol

Deployments

Ethereum: mainnet

File name	Contract deployed on mainnet	Comment
sUSX	0xbc4044...bbd2792d	proxy
sUSX	0xf3c976...dd69bd56	implementation

Base: mainnet

File name	Contract deployed on mainnet	Comment
sUSX	0xbc4044...bbd2792d	proxy
sUSX	0x9092fd...ab93a56e	implementation

Arbitrum: mainnet

File name	Contract deployed on mainnet	Comment
sUSX	0xbc4044...bbd2792d	proxy
sUSX	0x1c3b44...b0d5bc90	implementation

Optimism: mainnet

File name	Contract deployed on mainnet	Comment
sUSX	0xbc4044...bbd2792d	proxy
sUSX	0xe08020...f5035d75	implementation

BSC: mainnet

File name	Contract deployed on mainnet	Comment

File name	Contract deployed on mainnet	Comment
sUSX	0xbc4044...bbd2792d	proxy
sUSX	0xa4e5eb...c2d482cc	implementation

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	1
Medium	7
Low	12

ID	Name	Severity	Status
H-1	MintCap Check Missing in Cross-Chain Transfer	High	Fixed
M-1	Inaccurate `maxWithdraw` and `maxRedeem` Function Results	Medium	Fixed
M-2	Missing Function for Future Config Edits	Medium	Fixed
M-3	Potential Arbitrage Due to Rate Discrepancies in Cross-Chain Transfers	Medium	Acknowledged
M-4	Centralization Risks	Medium	Acknowledged
M-5	Potential Deposit Blockage	Medium	Fixed
M-6	Inability to perform a cross-chain transfer due to mintCap	Medium	Acknowledged
M-7	`deposit()` doesn't check future rewards from configured epochs	Medium	Acknowledged
L-1	`updateEpochId` gas optimization	Low	Fixed
L-2	Config range checks	Low	Fixed

L-3	The epoch configuration may overlap by 1 second	Low	Acknowledged
L-4	Unchecked in `for` loop is default behaviour since 0.8.22	Low	Acknowledged
L-5	Optimization Issues in Function `_rpow`	Low	Acknowledged
L-6	Inefficient Initializer Usage	Low	Acknowledged
L-7	Double `Ownable2Step` Initialization	Low	Fixed
L-8	Missing Zero-Address Checks	Low	Fixed
L-9	Incorrect Event Parameter in Function `_withdraw`	Low	Fixed
L-10	Incorrect Condition in Function `currentAPY`	Low	Fixed
L-11	Gas optimisation in `mul` and `rmul`	Low	Acknowledged
L-12	Blocking users' deposits by front-running	Low	Acknowledged

1.6 Conclusion

The project under review facilitates the cross-chain distribution of revenue among users in USX tokens. The owner determines the interest amounts to ensure that potential rewards do not exceed the allocated revenue for distribution.

The scope of the audit includes the following contracts:

- **sUSX** - An ERC4626 token with a fixed exchange rate which is calculated based on interest rates set in advance by the owner through USR configuration periods.
- **USR** - An abstract contract that manages the addition of new interest rate periods. These rates can be negative, encouraging users to withdraw their funds from sUSX.
- **RateMath** - A library that provides pure functions for safe multiplication and exponentiation with 27-digit precision.
- **ProxyAdmin2Step** - An OpenZeppelin ProxyAdmin contract modified with the Ownable2Step mechanism.

The sUSX contract is expected to be the minter in DForce's MSDController contract to mint and burn USX tokens, which is the MSD asset. The sUSX can be deployed on multiple chains where the MSDController and USX tokens are present. The number of tokens minted on any chain is limited by mintCap parameters, and the total of these mintCaps across networks defines the maximum revenue that can be minted. sUSX can perform cross-chain transfers using the LayerZero protocol, converting project revenue to USX rewards by buying back and burning USX tokens equal to or greater than the allocated amount for sUSX rewards.

The main attack vectors and concerns examined in the audit include:

1. **Cross-Chain Architecture**

   - sUSX tokens can be exchanged for USX tokens on any network they are present, which could lead to rate desynchronization and potential arbitrage opportunities. To mitigate this, implement monitoring systems to track rate synchronization across different chains or perform cross-chain transfers based on asset values instead of shares.
   - Incoming cross-chain transfers can be reverted if the mintCap is reached or if the sUSX project is paused, potentially locking user funds.

2. **Centralization Risks**

   - The stability of the project depends heavily on the owner's actions, as rate desynchronization could lead to arbitrage opportunities. These desynchronizations can also occur due to halts or pauses in side-chains, leading to discrepancies in added configurations across chains. We recommend using a MultiSig account for the owner role and setting up monitoring systems to prevent the vulnerability for the project activity. Additionally, implementing a function to allow the editing of planned interest rate configurations is advised.

3. **Collateralization Guarantees**

- The collateralization of generated interest must be greater than or equal to the sum of mintCaps set in MSDControllers across networks. A high-severity issue was identified, where the total minted value can exceed the sum of mintCaps through cross-chain transfers.
- If the generated rewards exceed the mintCap, not all users can withdraw their funds from sUSX until the admin increases the mintCap value and allocates new tokens for distribution.We recommend ensuring that reward amounts do not exceed mintCaps when adding new interest rate configurations and minting sUSX tokens.

4. **Math Operations**

- The RateMath contract was meticulously checked to affirm it correctly implements the algorithm for exponentiation. Assembly Yul instructions are used safely, without performing unsafe memory operations, and correctly handle boundary values.
- All operations are performed with RAY precision (10^27), which provides accurate calculations for exchange rates and reward distributions.

Implementing these recommendations and addressing the findings presented in the second section of this report will make the project more robust and resilient. To further enhance the project's integrity and reliability, the following actions are recommended:

- **Integrate Monitoring Mechanisms** to track exchange rate dynamics across different networks and cross-chain activities.
- **Conduct Periodic Independent Audits** of the project's codebase to provide objective assessments of its security posture and identify new vulnerabilities.
- **Create a Bug Bounty Program** to incentivize a broader security community to identify and report vulnerabilities, leveraging the expertise of individual researchers to enhance project security.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	MintCap Check Missing in Cross-Chain Transfer
Severity	High
Status	Fixed in d442b32e

Description

The issue was identified within the sUSX.sol#L276 function of the `sUSX` contract.

There is no check to ensure that the total mint amount does not exceed the `MSDController.mintCap` parameter. This oversight could lead to an attack vector where the mintCap limit is bypassed through cross-chain transfers. The following scenario outlines the potential attack:

1. `sUSX.totalMint` equals `MSDController.mintCap(sUSX)`, preventing direct USX withdrawals on the current network.
2. A cross-chain transfer is performed to a chain where the mintCap limit hasn't been reached.
3. The `unstakedAmount` on the sending network increases by the `assets` value, causing `sUSX.totalMint` to surpass the `MSDController.mintCap(sUSX)` value.
4. On the receiving chain, the `stakedAmount` increases by the `assets` amount.
5. Assets are withdrawn on the receiving chain, decreasing the `unstakedAmount` by the same value, thereby the `totalMint` value on the receiving chain remains unchanged.

This manipulation allows the number of minted USX token rewards to exceed the sum of mintCaps on all chains, potentially bypassing the collateralization limits of the `sUSX` interest.

The issue is classified as **high** severity as it can significantly weaken the collateralization guarantees of the `sUSX` interest.

Recommendation

We recommend implementing a check to ensure that the `totalMint()` value after `_burn` does not exceed the `MSDController.mintCap(sUSX)` in `outBoundTransferShares` function.

Client's commentary

> Fixed in ffd69d1c.

2.3 Medium

M-1	Inaccurate `maxWithdraw` and `maxRedeem` Function Results
Severity	Medium
Status	Fixed in d442b32e

Description

The issue was identified within the sUSX.sol contract.

These functions do not account for the `MSDController.mintCap` value, which leads to a discrepancy between the result of these view functions and the actual amount that users can redeem or withdraw. Although withdraws and redeems will revert if they exceed the `mintCap`, the view functions do not provide an accurate reflection of the real limits.

The issue is classified as **medium** severity as it can mislead users and potentially cause confusion and failed transactions.

Recommendation

We recommend updating the `maxWithdraw` and `maxRedeem` functions to include checks against the `MSDController.mintCap` value to provide a more precise and accurate representation of the maximum amounts that can be withdrawn or redeemed.

Client's commentary

> Fixed in 53a91618.

M-2	Missing Function for Future Config Edits
Severity	Medium
Status	Fixed in d442b32e

Description

The issue was identified within the USR.sol contract.

There is no function to edit the configuration for future USR settings. This omission could lead to discrepancies between the rates on different chains if a configuration is added mistakenly or with mistyped parameters. Having an edit function is necessary to ensure that any configuration changes are made correctly and efficiently before their start time.

The issue is classified as **medium** severity as it can lead to inconsistencies and potential operational issues across different chains.

Recommendation

We recommend adding a function to edit USR configurations, ensuring that the `startTime` of the configuration being edited is greater than the current block timestamp.

Client's commentary

> Fixed in f507817a
> Add `_updateLastEpochEndTime` and `_deleteLastEpoch` respectively.

M-3	Potential Arbitrage Due to Rate Discrepancies in Cross-Chain Transfers
Severity	Medium
Status	Acknowledged

Description

The issue was identified within the sUSX.sol#L276 and sUSX.sol#L286 functions of the `sUSX` contract.

Potential discrepancies in rates on different chains can lead to an arbitrage opportunity, enabling the following attack scenario:

1. An attacker mints `n` sUSX on the chain with a lower rate (`rate1`), burning `n * rate1` USX tokens.
2. The attacker transfers the `n` sUSX to a chain with a higher rate (`rate2`).
3. The attacker withdraws `n * rate2` USX tokens, gaining `n * (rate2 - rate1)` USX tokens.

This attack exploits rate discrepancies between chains, potentially draining the USX vaults on the chain with the a higher rate. Notably, the `totalSupply` and `totalMint` values on both chains remain unchanged, bypassing the mintCap limits.

The severity of this issue is classified as **medium**. While the potential for exploiting rate discrepancies can lead to significant asset drainage from the system, the likelihood of such an event occurring requires a prolonged period where significant differences in the rates configured by the owner across blockchain networks persist without corrective actions by the admin.

Recommendation

We recommend performing cross-chain transfers based on a fixed assets value rather than shares to ensure the value of the transferred tokens remains consistent across different chains, independent of the cross-chain rates.

Client's commentary

> sUSX is designed to share a global rate, will ensure identical rate config on all chains, the timestamp differeces are allowed.

M-4	Centralization Risks
Severity	Medium
Status	Acknowledged

Description

The issue was identified within the contract's ownership and role management structure, presenting several centralization risks:

1. Owners have the capability to upgrade the contract implementation.
2. The `PAUSER_ROLE` can pause all transfers of sUSX, including mints and burns, effectively halting withdrawal activities.
3. The `BRIDGER_ROLE` is capable of minting and burning new tokens, necessitating that this role be exclusively assigned to verified and audited contracts.
4. Owners are responsible for setting the interest rate configurations to ensure that the incoming rewards are collateralized by the treasury assets.
5. Owners must set valid mintCap values to ensure that rewards do not exceed these caps and that the caps are less than the collateral in the treasury.
6. Owners are responsible for synchronizing the exchangeRate across different chains to prevent arbitrage activities.

These centralization risks highlight the significant control owners have over the system's critical functions, which could be exploited if not managed properly.

The issue is classified as **medium** severity due to the potential for abuse of power, which can impact the system's integrity and users' trust.

Recommendation

We recommend implementing the following measures to mitigate centralization risks:

1. Ensure the owner is a multisig account to distribute control among multiple parties.
2. Assign the `BRIDGER_ROLE` exclusively to verified and audited contracts.
3. Establish valid configurations and procedures to guarantee the collateralization of rewards.
4. Ensure prompt actions can be taken to prevent rate desynchronization and potential arbitrage opportunities.

Client's commentary

> The owner is a multisig and will ensure the correctness of all privileged actions.

M-5	Potential Deposit Blockage
Severity	Medium
Status	Fixed in d442b32e

Description

The `maxDeposit` sUSX.sol#L229 will attempt to calculate the maximum amount that can be deposited based on the new `mintCap`, but it will revert due to underflow because `mintCap - totalSupply` is negative. This situation is not triggered by user actions but can occur if the admin changes the `mintCap` value.

This issue is marked as **medium** because owner actions can lead to a temporary contract lock.

Recommendation

We recommend avoiding reverting by returing 0 in maxDeposit.

Client's commentary

> Fixed in 9a97c8b9.

M-6	Inability to perform a cross-chain transfer due to mintCap
Severity	Medium
Status	Acknowledged

Description

Cross-chain transfers in the sUSX contract may be reverted on the destination chain in sUSX.sol#L290 if the mint cap (`mintCap`) is reached. This issue arises because different chains may have different mintCap settings. When the mint cap on the destination chain is reached, the transaction reverts, leading to the following problems:

- Funds Locked: User's funds get locked, preventing the completion of the withdrawal.
- Admin Intervention Required: The only way to resolve this issue is for the administrator to increase the mintCap on the destination chain.
- No Option to Withdraw Funds Back: Users have no ability to withdraw their funds back to the source chain if the transfer fails due to the mint cap limit.

This issue is marked as **medium** because it results in the temporary locking of users' funds in case of incorrect configuration.

Recommendation

We recommend implementing checks and fallback mechanisms to handle such scenarios.

Client's commentary

> Admin can check the validity and intervene, or it can be resolved as others leaving the protocol on destination chain.

M-7	`deposit()` doesn't check future rewards from configured epochs
Severity	Medium
Status	Acknowledged

Description

The deposit() function in the `sUSX` contract currently only checks the mintCap for `sUSX` tokens and does not account for the future rewards that will be generated from the configured epochs (USRConfigs). This can lead to a scenario where, upon withdrawal, users might face a mintCap limit for the USX token. Specifically, the total amount of USX that needs to be minted (including both the principal and the accrued rewards) may exceed the mintCap, resulting in a failed withdrawal.

This issue marked as **medium** because the owner actions result in the temporary locking of users' funds.

Related code: sUSX.sol#L238

Recommendation

We recommend including a future rewards amount to the `mintCap` limit check.

Client's commentary

> Ensure re-estimating `mintCap` and changing it if necessary when adjusting the sUSX reward.

2.4 Low

L-1	`updateEpochId` gas optimization
Severity	Low
Status	Fixed in d442b32e

Description

The `updateEpochId` modifier is intended to keep the `lastEpochId` value up to date, modifying it on demand. This value is changed rarely; however, the current implementation updates it every time it is called. This results in more gas being spent on the SSTOR EVM opcode.

Related code: USR.sol#L39

Recommendation

We recommend optimizing the code to avoid unnecessary gas spending.

Client's commentary

> Fixed in d442b32e.

L-2	Config range checks
Severity	Low
Status	Fixed in d442b32e

Description

Considering that the epoch configuration cannot be changed once it has been set, an erroneously long epoch duration can become a problem requiring a contract redeployment. Such an error can occur, for example, if the administrator mistakenly sets the epoch size 1000 times larger by specifying the interval in milliseconds instead of seconds.

Related code: USR.sol#L101

Recommendation

We recommend checking that the epoch duration is not abnormally large and prohibiting the addition of such epochs.

Client's commentary

> Also fixed in f507817a, as it allows editing and deleting configs.

L-3	The epoch configuration may overlap by 1 second
Severity	Low
Status	Acknowledged

Description

When adding a new epoch configuration, it is checked that the new epoch does not overlap with the previous one. However, the current implementation allows for a one-second overlap, meaning that the last second of one epoch coincides with the first second of the next epoch. This can potentially cause issues with logic.

Related code: USR.sol#L113

Recommendation

We recommend prohibiting any overlap of epochs, even by one second.

Client's commentary

> The overlap of a specific timestamp is allowed as no interest accured at that point.

L-4	Unchecked in `for` loop is default behaviour since 0.8.22
Severity	Low
Status	Acknowledged

Description

Starting from Solidity compiler version 0.8.22, the use of unchecked for iterating variables in loops is no longer necessary. The compiler now includes optimizations that automatically handle overflow checks for these variables. Previously, developers had to explicitly use the `unchecked` block to bypass these checks and save gas. However, the improvements in version 0.8.22, allow the compiler to manage this efficiently by itself, simplifying the code and reducing the chance of errors.

Related code: USR.sol#L155

Recommendation

We recommend upgrading to the latest Solidity compiler version to take advantages of its new features.

Client's commentary

> Acknowledged.

L-5	Optimization Issues in Function `_rpow`
Severity	Low
Status	Acknowledged

Description

The issue was identified within the RateMath.sol#L7 function of the `RateMath` contract.

Several optimization opportunities were found within the assembly code:

1. `mod(_, 2)` is equivalent to `and(_, 1)` and takes less gas.
2. `div(_, 2)` is equivalent to `shr(_, 1)` and takes less gas.
3. The `div` operation in Yul returns 0 if the second parameter is 0, rendering the `iszero(iszero(x))` check redundant.
4. The `base` argument can be removed, and the `RAY` constant used instead, as all the calls of this function are performed with the `RAY` parameter.

The issue is classified as **low** severity as it does not pose significant risks but can lead to unnecessary gas consumption and complexity.

Recommendation

We recommend the following optimizations:

1. Replace `mod(_, 2)` with `and(_, 1)`.
2. Replace `div(_, 2)` with `shr(_, 1)`.
3. Remove the redundant `iszero(iszero(x))` check.
4. Remove the `base` argument and use the `RAY` constant directly.

Client's commentary

> Acknowledged.

L-6	Inefficient Initializer Usage
Severity	Low
Status	Acknowledged

Description

The issue was identified within the sUSX.sol#L58 of the `sUSX` contract.

The constructor uses the `initialize` function, which is not the optimal approach for preventing reinitialization attacks. Instead, the `_disableInitializers()` function should be used to ensure the contract cannot be reinitialized.

The issue is classified as **low** severity as it does not pose a significant security risk but can lead to potential misuse if not properly addressed.

Recommendation

We recommend using the `_disableInitializers()` function in the constructor instead of the `initialize` function to prevent any potential reinitialization.

Client's commentary

> Acknowledged.

L-7	Double `Ownable2Step` Initialization
Severity	Low
Status	Fixed in d442b32e

Description

The issue was identified within the sUSX.sol#L82 function of the `sUSX` contract.

The function performs double initialization of `Ownable2Step`, resulting in two events being emitted, with the second one inside the `__USR_init` function. This redundancy can lead to unnecessary gas consumption and potential confusion in event logs.

The issue is classified as **low** severity as it does not pose a significant risk but can lead to inefficiency and confusion.

Recommendation

We recommend removing the redundant initialization of `Ownable2Step` to avoid emitting duplicate events and to improve the efficiency of the contract.

Client's commentary

> Fixed in d442b32e.

L-8	Missing Zero-Address Checks
Severity	Low
Status	Fixed in d442b32e

Description

The issue was identified within the sUSX.sol#L74-L75 function of the `sUSX` contract.

There are no checks for zero-address values for the `_usx` and `_msdController` parameters. This oversight can potentially disrupt the functionality of the contract if these parameters are mistakenly set to zero.

The issue is classified as **low** severity as it can be easily exploited if not addressed, but it does not directly lead to security vulnerabilities.

Recommendation

We recommend adding checks to ensure that the `_usx` and `_msdController` parameters are not set to zero addresses.

Client's commentary

> Fixed in d442b32e.

L-9	Incorrect Event Parameter in Function `_withdraw`
Severity	Low
Status	Fixed in d442b32e

Description

The issue was identified within the sUSX.sol#L203 function of the `sUSX` contract.

The function emits a `Withdraw` event using `msg.sender` as the caller parameter instead of the `caller` variable. This can lead to incorrect event data, which may cause confusion and inaccuracies in event tracking and monitoring.

The issue is classified as **low** severity as it does not directly impact the contract's core functionality but affects the accuracy of emitted events.

Recommendation

We recommend modifying the `Withdraw` event to use the `caller` variable instead of `msg.sender` to ensure accurate event logging.

Client's commentary

> Fixed in d442b32e.

L-10	Incorrect Condition in Function `currentAPY`
Severity	Low
Status	Fixed in d442b32e

Description

The issue was identified within the USR.sol#L196-L199 function of the `USR` contract.

The condition checking if `block.timestamp` falls within the `startTime` and `endTime` of the current USR configuration is incorrect. It uses `>` and `<` signs, which causes the `currentAPY` function to return 0 for the exact `startTime` and `endTime`. This can mislead users, as it it would be more accurate to return the correct APY values at the boundaries, which also inform users that the period is about to start or end.

The issue is classified as **low** severity as it does not pose significant risks but can cause minor inaccuracies and confusion.

Recommendation

We recommend rewriting the condition using `>=` and `<=` signs to ensure that the function returns non-zero values for the exact `startTime` and `endTime`. This will clearly signal to users that the APY period is starting or ending.

Client's commentary

> Fixed in d442b32e.

L-11	Gas optimisation in `mul` and `rmul`
Severity	Low
Status	Acknowledged

Description

The `_rmul` function can be optimized:

```
        assembly {
            let prod := mul(x, y)
            if iszero(eq(div(prod, x), y)) {
                revert(0, 0)
            }
            z := div(prod, RAY)
        }
```

The `_mul` function optimization

```
        assembly {
            z := mul(x, y)
            if iszero(eq(div(z, x), y)) {
                revert(0, 0)
            }
        }
```

This issue is marked as **low** because it impacts contract optimization rather than security.

Related code: RateMath.sol

Recommendation

We recommend using an optimized version of the functions to safe gas.

Client's commentary

> Acknowledged.

L-12	Blocking users' deposits by front-running
Severity	Low
Status	Acknowledged

Description

An attacker can monitor the transaction pool (mempool) and see upcoming deposit transactions. The attacker can then submit their own transaction with a higher gas fee to be processed before the victim's transaction. As a result, the victim's transaction will be reverted by the cap limit control check.

Related code: sUSX.sol#L235

Recommendation

We recommend implementing an exit fee to make this attack less attractive to the attacker.

Client's commentary

> Acknowledged.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts


https://github.com/mixbytes/audits_public


https://mixbytes.io/


hello@mixbytes.io


https://twitter.com/mixbytes