DFORCE LENDING V2 SECURITY AUDIT REPORT

Jun 25, 2024

MixBytes()

TABLE OF CONTENTS

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

The dForce project is a decentralized lending protocol. Users can supply collateral, borrow assets and earn interest. The features of the dForce protocol are:

- SuperCharged mode - categorizing some assets by having corellated prices to allow less collaterization ratios;
- liquidation threshold - a buffer between the maximum borrowing power and insolvency
- Segregated mode - limit risks of collaterization by some assets;
- delay payment - timelock on transfer-outs if some conditions are met.

1.4 Project Dashboard

Project Summary

Title	Description
Client	dForce
Project name	Lending V2
Timeline	06.09.2023 - 17.06.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
06.09.2023	6f3a7b6946d8226b38e7f0d67a50e68a28fe5165	Initial commit for the audit
14.11.2023	abf7ef8d327a15a9e5e5f8bec6b444142d988f34	Commit for the re-audit
29.11.2023	490a30f5a2e0e369f9ea52097b28254f11c5ada6	Commit for the re-audit 2
13.06.2024	5d005d16a96499828a6703f41cda2b946887800e	Commit for the diff audit

Project Scope

The audit covered the following files:

File name	Link
Controller.sol	Controller.sol
ControllerStorage.sol	ControllerStorage.sol

File name	Link
ControllerV2ExtraBase.sol	ControllerV2ExtraBase.sol
ControllerV2ExtraExplicit.sol	ControllerV2ExtraExplicit.sol
ControllerV2ExtraImplicit.sol	(ControllerV2ExtraImplicit.sol
ControllerV2.sol	ControllerV2.sol
DefaultTimeLock.sol	DefaultTimeLock.sol
iETH.sol	iETH.sol
iETHV2.sol	iETHV2.sol
iToken.sol	iToken.sol
iTokenV2.sol	iTokenV2.sol
RewardDistributorSecondV3.sol	RewardDistributorSecondV3.sol
RewardDistributorV3.sol	RewardDistributorV3.sol
TimeLockStrategy.sol	TimeLockStrategy.sol
TokenBase/Base.sol	Base.sol
TokenBase/InterestUnit.sol	InterestUnit.sol
TokenBase/TokenAdmin.sol	TokenAdmin.sol
TokenBase/TokenERC20.sol	TokenERC20.sol
TokenBase/TokenEvent.sol	TokenEvent.sol
TokenBase/TokenStorage.sol	TokenStorage.sol

Deployments on Base:mainnet

File name	Contract deployed	Comment
Timelock	0xa4e5ebEdcD1129Ed30C77644a70F4dd3c2d482cc	
ControllerV2	0xBae8d153331129EB40E390A7Dd485363135fcE22	proxy
ControllerV2	0xc7d598e4434d51273bbb0418a9e764b53ddc7d63	implementation
ControllerV2ExtraImplicit	0x95c06b4b6902b0aE37dDFf281d2c785313C86691	
ControllerV2ExtraExplicit	0xd556fb139a36F5EA809636B9858C5e3fe1613EA4	
DefaultTimeLock	0xD614E4a3C1152812Da43E824930376AA8b7D8B1d	proxy
DefaultTimeLock	0x28bbd52b8e6b46210fcdf6d605e242a72240ef66	implementation
TimeLockStrategy	0x4ca6A624808a7B248238c138558CA3047d9E2E3F	proxy
TimeLockStrategy	0xeae18e7d342d03a5fb0492060557f979b9e9a92f	implementation
iETHV2	0x76B5f31A3A6048A437AfD86be6E1a40888Dc8Bba	proxy
iETHV2	0x0d66fa17fac4b7d35240ff58d278ddd2f036451f	implementation
iTokenV2 iwstETH	0xf8fBD6202FBcfC607E31A99300e6c84C2645902f	proxy
iTokenV2 icbETH	0x6D9Ce334C2cc6b80a4cddf9aEA6D3F4683cf4a50	proxy
iTokenV2 iUSDC	0xBb81632e9e1Fb675dB5e5a5ff66f16E822c9a2FD	proxy
iTokenV2 iUSX	0x82AFc965E4E18009DD8d5AF05cfAa99bF0E605df	proxy
iTokenV2	0x66d7c93c22935d436dfb1e85394ae52c0a2be001	implementation
rewardDistributorProxy	0xE08020a6517c1AD321D47c45Efbe1d76F5035d75	proxy

File name	Contract deployed	Comment
RewardDistributorSecondV3	0x251687dd69ceae80f9f2b384e7a7c6a58a4dff7d	implementation

1.5 Summary of findings

Severity	# of Findings
Critical	2
High	5
Medium	7
Low	11

ID	Name	Severity	Status
C-1	Inflation attack on iToken	Critical	Fixed
C-2	A detached reward distributor can be drained	Critical	Fixed
H-1	The `ControllerV2` implementation can be destroyed by an attacker	High	Fixed
H-2	Manipulation of global daily limits on `TimeLockStrategy`	High	Acknowledged
H-3	Funds may freeze on the `TimeLock` if the beneficiary does not implement `claim()`	High	Fixed
H-4	Tokens in Segregated mode cannot be fully repaid by borrowers	High	Fixed
H-5	Seizing assets as collateral without entering the market may result in incorrect value calculation	High	Fixed
M-1	`_exitMarket` always returns `true`, even on error	Medium	Fixed
M-2	Lack of speed-up functionality in the `TimeLock`	Medium	Fixed

M-3	Potential desynchronization between asset transfer and agreement creation in the `TimeLock`	Medium	Acknowledged
M-4	The lack of support of fee on transfer tokens in `DefaultTimeLock`	Medium	Acknowledged
M-5	Changing the `rewardToken` during distribution in `RewardDistributor` is dangerous	Medium	Acknowledged
M-6	Vulnerabilities to rug pull scenarios	Medium	Acknowledged
M-7	Assets may be unexpectedly seized	Medium	Acknowledged
L-1	`isController` reports `true` on the implementation contract	Low	Acknowledged
L-2	`extraImplicit` and `extraExplicit` are declared twice	Low	Fixed
L-3	A redundant `market` parameter in `exitMarketFromiToken`	Low	Acknowledged
L-4	A misleading function name `unfreezeAllAgreements`	Low	Fixed
L-5	The lack of verification of `timeLock.controller` in `_setTimeLock` setter	Low	Fixed
L-6	Missing validations for non-zero `mintAmount`, `borrowAmount` and `repayAmount`	Low	Acknowledged
L-7	Permit logic doesn't follow the ERC-2612 specification	Low	Acknowledged
L-8	The Solidity version is not up to date	Low	Acknowledged
L-9	Unintended ETH `receive` in the Controller	Low	Fixed
L-10	Using OpenZeppelin `__disableInitializers` in ControllerV2ExtraBase	Low	Acknowledged
L-11	Using the OpenZeppelin `EnumerableSetUpgradeable.values()` function	Low	Acknowledged

1.6 Conclusion

The project encountered well-known issues such as inflation attack and proxy implementation self-destruction, which according to the developers were known to them and were supposed to be addressed through the correct deployment procedure. We recommend always resolving such issues through the code of smart contracts.

We also recommend enhancing the test coverage by better evaluating both positive and negative scenarios in the behavior of functions.

It is also important to remember that the user of the system can be not only an EOA (Externally Owned Account) but also a smart contract, which imposes certain limitations on the user's ability to interact with the system.

Dividing the Controller code into several smart contracts with non-trivial mutual calls complicates reading and analyzing the code. To enhance security, we recommend using simpler architectural solutions whenever possible.

During the audit, 2 critical, 5 high, 7 medium and 11 low severity issues have been discovered. All issues are confirmed by the developers and fixed or acknowledged.

2. FINDINGS REPORT

2.1 Critical

C-1	Inflation attack on iToken
Severity	Critical
Status	Fixed in ebeee963

Description

Until `iToken` has sufficient `totalSupply`, an attacker can manipulate the `underlying`/`iToken` exchange rate by directly transferring the underlying asset to the `iToken` smart contract. This leads to rounding issues in `mint` and `redeemUnderlying` causing a user to lose some amount of their underlying assets.

Due to the possibility of permanent loss of user assets, such issues have a critical severity rating.

Related code:

- rounding issues on mint
  Base.sol#L199
- rounding issues on redeem underlying in iToken for native token
  iETH.sol#L140
- rounding issues on redeem underlying in iToken for ERC20 iToken.sol#L126

Recommendation

Although this issue can be hotfixed through accurate deployment procedures and configuration settings, we recommend fixing it at the smart contract code level either by preventing the iToken from having a nonzero but small `totalSupply` or by ensuring accurate accounting of the underlying asset in the smart contract.

C-2	A detached reward distributor can be drained
Severity	Critical
Status	Fixed in 490a30f5

Description

If admin decided to change the current reward distribution logic and set new one by using the Controller.sol#L548 function,
the prev version is supposed to distribute rewards for the prev period.
After detaching the reward distribution contract from the controller, transfers don't track by the controller any more and by abusing this issue an attacker can drain rewards from the old distributor by using cycles charge balance then claim from different accounts or a flashloan attack.

Recommendation

We recommend following one of the two ways:

- allow tracking transfers by a few distributors at the same time,
- don't change the distributor address and use migration.

2.2 High

H-1	The `ControllerV2` implementation can be destroyed by an attacker
Severity	High
Status	Fixed in bf28390f

Description

The `ControllerV2` implementation code is vulnerable to a direct call of `initialize`. Since `initialize` executes `delegatecall` to an arbitrary address, an attacker can destroy the Controller's implementation contract, thus freezing the entire system until manual intervention by the proxy administrator occurs. This is accordingly rated as high in severity.

Related code - `delegatecall` to the arbitrary address: ControllerV2.sol#L57

Recommendation

Although this vulnerability can be hotfixed through an accurate deployment process, we recommend addressing it at the smart contract code level by preventing direct calls to `initialize` against the implementation address.

H-2	Manipulation of global daily limits on `TimeLockStrategy`
Severity	High
Status	Acknowledged

Description

The global daily limits implemented in the audited code are susceptible to manipulation by an attacker, leading to inconvenience for legitimate users due to the time lock on any outgoing transfers. Since the system will remain in an undesired state until the smart contract owner intervenes, this issue is rated as high in severity.

Related code - procedure for accumulating daily statistics:
TimeLockStrategy.sol#L166

Recommendation

We recommend reworking the global limits to prevent manipulation.

Client's commentary

> We are aware of this, and working on a more sophisticated strategy to decide the delay of a outgoing transfer.

H-3	Funds may freeze on the `TimeLock` if the beneficiary does not implement `claim()`
Severity	High
Status	Fixed in 8ad82e8e

Description

Assets from the `TimeLock` can only be claimed by their respective beneficiaries via calling the `claim` function. However, if the beneficiary is an immutable smart contract with no ability to invoke `claim` against the `TimeLock`, the locked assets become inaccessible to the beneficiary. Given that some accounts will be unable to access their assets until the manual intervention of the smart contract owner, this issue is rated as high in severity.

Related code - procedure of agreement execution:
DefaultTimeLock.sol#L81

Recommendation

We recommend allowing any account to invoke the `claim`.

H-4	Tokens in Segregated mode cannot be fully repaid by borrowers
Severity	High
Status	Fixed in 820d9182

Description

Tokens that have the Segregated mode activated possess a designated `MarketV2.currentDebt` value. This value is prevented from surpassing the `debtCeiling` through borrow functions. Notably, the `ControllerV2ExtraExplicit.afterRepayBorrow` function employs the `SafeMath.sub` function to subtract the amount of repaid underlying assets from the `currentDebt` value. This function is designed to revert any underflow errors. However, the `currentDebt` value does not consider that the debt is increasing over time with `InterestRateModel`, associated with the `iToken`. Consequently, the repaid amount always exceeds the borrowed sum, causing borrowers unable to fully repay their debt until the contract's owner updates the `ControllerV2ExtraExplicit` implementation.

This issue is labeled as `high`, since it imposes the potential to temporarily block specific `repayBorrow` transactions.

Related code - `beforeBorrow` for Segregated mode: ControllerV2ExtraExplicit.sol#L200

Recommendation

We recommend reseting the `currentDebt` value to `zero` in cases where `currentDebt` is less than `repayAmount`.

H-5	Seizing assets as collateral without entering the market may result in incorrect value calculation
Severity	High
Status	Fixed in fa5cfaf1

Description

During liquidation, collateral may be seized even if the borrower has not entered the market with it. Sanity checks regarding the price oracle status for the seized asset will be skipped if the market has not been entered for this asset.

This issue is labeled as `high` since an outdated or inaccurate `iTokenCollateral` price could result in either excessive or insufficient payments to the liquidator.

Related code:

- the `liquidateCalculateSeizeTokensV2` function: ControllerV2ExtraImplicit.sol#L477
- `_liquidateBorrowInternal` iTokenV2.sol#L76
- `beforeLiquidateBorrow` ControllerV2.sol#L346

Recommendation

We recommend prohibiting the seizure of assets that are not explicitly listed by the borrower as allowed collateral through `enterMarket`.

2.3 Medium

M-1	`_exitMarket` always returns `true`, even on error
Severity	Medium
Status	Fixed in 6315b9a2

Description

The `_exitMarket` function, as per its specification, is designed to return `false` if the market isn't listed or not entered. However, in its current implementation, the function always returns `true`, leading to inconsistency between the expected and actual outcomes.

Related code:

- function returns `true` even if the token is not listed Controller.sol#L1401
- function returns `true` even if the market is not entered Controller.sol#L1406

Recommendation

We recommend adjusting the `_exitMarket` function to return values in accordance with the expectations of both users and developers as well as the specification.

M-2	Lack of speed-up functionality in the `TimeLock`
Severity	Medium
Status	Fixed in 1df8a1da

Description

Once created, an agreement in the `TimeLock` enforces a delay until the expiration time specified during the agreement's creation. If the delays are unintentionally long, the only remedy is to replace the `TimeLock` implementation.

Related code - procedure of agreement execution: DefaultTimeLock.sol#L83

Recommendation

We recommend implementing speed-up functionality in the `TimeLock` to address unintentionally prolonged delays.

M-3	Potential desynchronization between asset transfer and agreement creation in the `TimeLock`
Severity	Medium
Status	Acknowledged

Description

In the audited code, asset transfer and agreement creation are treated as two separate processes.

- Assets can be transferred to the `TimeLock` without creating an agreement, leading to them being frozen.
- An agreement can be created without transferring assets and may be satisfied using assets intended for other agreements, rendering those agreements unsatisfiable.

Related code - agreement creation: DefaultTimeLock.sol#L47

Recommendation

Although the asset transfer and the agreement creation are currently synchronized (outside of the `TimeLock` smart contract), we recommend synchronizing them within the `TimeLock` smart contract itself to maintain the `TimeLock` state consistency.

Client's commentary

> Such solution is chosen as it simplifies the logic of `iToken`'s outgoing transfer and does not require additional `approve` to `TimeLock`.

M-4	The lack of support of fee on transfer tokens in `DefaultTimeLock`
Severity	Medium
Status	Acknowledged

Description

The agreement creation precedes the token transfer. If a fee on transfer tokens is used, then the amount of tokens transferred may be reduced (due to transfer fees) and become less than the amount specified in the agreement for claiming.

This issue is labeled as `medium` since the resulted inconcistency can block the `claim` function until the `balance` of the timelock surpasses the quantity of tokens noted in the agreement.

Related code - agreement creation: DefaultTimeLock.sol#L47

Recommendation

We recommend reworking the `TimeLock` architecture to pull assets by `transferFrom` in the `createAgreement`. Additionally, it will help addressing previous finding.

Client's commentary

> We are aware of it and will carefully choose assets onboarding.

M-5	Changing the `rewardToken` during distribution in `RewardDistributor` is dangerous
Severity	Medium
Status	Acknowledged

Description

Alterations to the `rewardToken` in the middle of distribution, especially without verifying the congruence of `decimals` between the previous and the new token and ensuring price consistency, can lead to potential risks of excessive or insufficient rewards to distribution recipients.

Related code - procedure of updating the reward token: RewardDistributorV3.sol#L105

Recommendation

We recommend disabling the `_setRewardToken` function if the current `rewardToken` is a non-zero address.

Client's commentary

> `_setRewardToken` will normally only be set once, we prefer to keep some flexibility here.

M-6	Vulnerabilities to rug pull scenarios
Severity	Medium
Status	Acknowledged

Description

The contracts are `Ownable` with a possibility to change the contracts implementation to arbitrary code. Also, some contracts have functions to retreive the `ERC-20` tokens by the owner (e.g. `RewardDistributorV3.rescueTokens`, `iToken._withdrawReserves`).

Recommendation

To minimize the risk of a rug pull, we recommend utilizing the MultiSig and TimeLock techniques as the owner to ensure that no single entity has unilateral control. In the long run, consider transitioning to a DAO for governance functions.

Client's commentary

> Currently the ownership is ultimately controlled by a MultiSig and the governance process can be found on https://snapshot.org/#/dforcenet.eth.

M-7	Assets may be unexpectedly seized
Severity	Medium
Status	Acknowledged

Description

During liquidation, collateral may be seized even if the borrower has not entered the market with it. Such behavior is likely unexpected for the borrower.

Related code:

- the `liquidateCalculateSeizeTokensV2` function: ControllerV2ExtraImplicit.sol#L477
- `_liquidateBorrowInternal` iTokenV2.sol#L76
- `beforeLiquidateBorrow` ControllerV2.sol#L346

Recommendation

We recommend prohibiting the seizure of assets that are not explicitly listed by the borrower as allowed collateral through `enterMarket`.

Client's commentary

> It is a *feature* to ensure the protocol's solvency.

2.4 Low

L-1	`isController` reports `true` on the implementation contract
Severity	Low
Status	Acknowledged

Description

The `isController` view function is designed to prevent the accidental specification of an incorrect smart contract address as the controller address. However, the address of the `Controller` implementation incorrectly returns `true`, even though the valid controller address is intended to be a proxy address, not the implementation address.

Related code - `isController` view function: Controller.sol#L60

Recommendation

To enhance sanity checks, we recommend ensuring `isController` returns `false` when called against the implementation address.

Client's commentary

> We prefer to keep the proxy a pure proxy, and in some scenarios (mostly test cases), the controller are non-proxied.

L-2	`extraImplicit` and `extraExplicit` are declared twice
Severity	Low
Status	Fixed in 27e7df6a

Description

The storage variables `extraImplicit` and `extraExplicit` are declared twice in the code:

- ControllerV2.sol#L39-L43
- ControllerStorage.sol#L221-L225

  This could potentially lead to unexpected behavior.

Recommendation

We recommend removing the redundant declaration in `ControllerV2.sol`.

L-3	A redundant `market` parameter in `exitMarketFromiToken`
Severity	Low
Status	Acknowledged

Description

The `exitMarketFromiToken` function is designed to let the `iToken` request an exit from the market for a specified account using the given `iToken`. However, by providing a market parameter different from `address(this)`, the iToken can be permitted to exit from a market other than its own.

Related code - exit from an arbitrary market: ControllerV2.sol#L512

Recommendation

We recommend eliminating the `market` parameter and utilizing `msg.sender` as a secure substitute.

Client's commentary

> `exitMarketFromiToken` is the conterpart of `enterMarketFromiToken`, which can be called from a `iTokenB` to collateralize `iTokenC` by a modified version(iMSDMiniPool.sol#L200).
> We prefer to keep the interface consistant.

L-4	A misleading function name `unfreezeAllAgreements`
Severity	Low
Status	Fixed in dd99b2ab

Description

Despite its name, the `unfreezeAllAgreements` function does not actually unfreeze all agreements. It merely removes the global freeze flag that applies to all agreements, but an agreement will remain frozen if it was previously frozen by `freezeAgreements`.

Related code - procedure of agreement execution: DefaultTimeLock.sol#L86

Recommendation

We recommend changing the name of the `unfreezeAllAgreements` function to one that is more indicative of its actual functionality.

L-5	The lack of verification of `timeLock.controller` in `_setTimeLock` setter
Severity	Low
Status	Fixed in 31d2a460

Description

The `_setTimeLock` setter does not perform verification whether `timeLock.controller` is equivalent to `address(this)` or not. This may lead to all the `transferOut` function for `iTokens` becoming inaccessible.

Related code - `_setTimeLock`: ControllerV2ExtraImplicit.sol#L57

Recommendation

We recommend ensuring the equivalence of `timeLock.controller` and `address(this)` within the `_setTimeLock` function.

L-6	Missing validations for non-zero `mintAmount`, `borrowAmount` and `repayAmount`
Severity	Low
Status	Acknowledged

Description

In the current codebase, validations ensuring that `mintAmount`, `borrowAmount` and `repayAmount` are greater than zero are absent in the `iToken.mint`,`iToken.borrow`, `iToken.repayBorrow` functions respectively.

These missing checks can lead to unintended consequences, such as misleading event emissions or and registering empty collateral or borrow assets to users.

Related code:

- `mintInternal` Base.sol#L179
- `borrowInternal` Base.sol#L261
- `repayInternal` Base.sol#L299

Recommendation

We recommend inserting validation checks ensuring that the amounts are greater than zero to the following functions: `Base.borrowInternal`, `Base.repayInternal`, `Base.mintInternal`.

Client's commentary

> Prefer to keep it as it is.

L-7	Permit logic doesn't follow the ERC-2612 specification
Severity	Low
Status	Acknowledged

Description

The `iToken` uses a non-standard way to implement `permit`. It may cause compatibility issues when used in a third-party project.

Related code - implementation of `permit` in the `iToken`: Base.sol#L524

Recommendation

We recommend using the way that OpenZeppelin recommends (ERC20Permit.sol).

Client's commentary

> Prefer to keep it as it is.

L-8	The Solidity version is not up to date
Severity	Low
Status	Acknowledged

Description

The modern major version of the Solidity compiler is 0.8, but most of the codebase uses version 0.6.12.

Recommendation

We recommend using the up to date Solidity version.

Client's commentary

> Prefer to keep it as it is.

L-9	Unintended ETH `receive` in the Controller
Severity	Low
Status	Fixed in 6845977f

Description

At line ControllerV2.sol#L167
there is a receiver declared that isn't used anywhere.

Recommendation

We recommend removing the unused `receive()` function to prevent sending ETH to the contract.

L-10	Using OpenZeppelin `__disableInitializers` in ControllerV2ExtraBase
Severity	Low
Status	Acknowledged

Description

To avoid the ability to directly call the initialize() function at the implementation contract address, the constructor currently calls the initialize() function.

```
constructor() public {
    __initialize();
}

function __initialize() internal initializer {
    __Ownable_init();
}
```

OpenZeppelin provides a special function intended to disable initializers from the constructor.

Recommendation

We recommend using the OpenZeppelin `_disableInitializers` function.

Client's commentary

> We've chosen to maintain our current usage due to the absence of this implementation in OpenZeppelin version 3.3.0, ensuring consistency within the project.

L-11	Using the OpenZeppelin `EnumerableSetUpgradeable.values()` function
Severity	Low
Status	Acknowledged

Description

Currently, a loop is used to retrieve the values of the `EnumerableSetUpgradeable`. However, there is a special function intended to retrieve the values of the `EnumerableSetUpgradeable`.

Recommendation

We recommend using the `values` function to retreive the values of the `EnumerableSetUpgradeable`.

Client's commentary

> We've chosen to maintain our current usage due to the absence of this implementation in OpenZeppelin version 3.3.0, ensuring consistency within the project.

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