PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For Aquarius Loan (Full)

16 November 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Aquarius Loan on the BitTorrent Chain. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Aquarius Loan is a fork of Compound Finance with some customized changes made. In the report, we have sectioned the various parts of the protocol into subsections, documenting the functionality and issues found in.

The contracts were deployed before our audit, and after the audit the following changes were made:

- `Comptroller` was redeployed, and the Unitroller implementation was changed to the new `Comptroller`
- `Atoken` was redeployed, and the implementation of all Atokens were changed to the new contract, with the exception of `CEther`
- `Oracle` was redeployed and the one used in the `Comptroller` was changed to the new contract

Do note that if the project decides to upgrade contracts in future, there is a risk of storage collision if they incorrectly deploy the implementation storage.

1.1 Summary

Project Name	Aquarius Loan
URL	https://www.aquarius.loan/
Network	BitTorrent Chain
Language	Solidity

1.2 Contracts Assessed

There are several contracts in the GitHub repository provided by the team that will not be used in production, and thus not included in the scope of audit. The contracts are listed below.

- `AErc20Immutable`
- `LegacyInterestRateModel`
- `CDaiDelegate`
- `LegacyInterestRateModel`
- `LegacyJumpRateModelV2`
- `DAIInterestRateModelV3`
- `WhitePaperInterestRateModel`
- `SimplePriceOracle`
- `AquariusAggregatorPriceOracle`
- `AggregatorV2V3Interface`
- `GovernorAlpha`
- `Timelock`

It should also be noted that `SimplePriceOracle` should never be used in any way, shape or form in production since anyone can set prices.

Name	Contract	Live Code Match
AErc20	aBNB: 0x8fA51C0F16389C9154da0FFf01f607e472D8eFBA aBTC_B: 0xDC8f714d9a5c67266d6da2E5d3267d1587Ea5710 aETH: 0xEAC09CB245988a0805ab61beeDc78c1c848c15F9 aTRX: 0x25CA01DaefC58b039A5aD97243d6FfE6899710Fc aUSDT: 0x03ef96F537A7CDa4411C8643afD9d8814D5B4906 aUSDC: 0xf050cFBfedE31A71D1C7b506d9CcE9E3C73eF7E0 aBUSD: 0x6c09AfdAB1DBdE96e2A869adB651D8203B1136E5 aUSDD: 0x87bf3aB1C752a4B12Fa114E10732d00D62e6519f Implementation: 0x97A5710c4a8c1113060Bd9c9Ad9b1943f304AB59	MATCH
AErc20Immutable	Dependency	MATCH
AToken	Dependency	MATCH
AErc20Delegate	0x97A5710c4a8c1113060Bd9c9Ad9b1943f304AB59	MATCH
AErc20Delegator	Dependency	MATCH
ATokenInterfaces	Dependency	MATCH
CEther	0x13F9A7F33550FE935BF7B544E536584d30eCf50a	MATCH
Comptroller	0xb5D23B66eb2208c897acea71eA8C5bc4ee026417	MATCH
ComptrollerInterface	Dependency	MATCH
ComptrollerStorage	Dependency	MATCH
Unitroller	Proxy: 0x6056Eb6a5634468647B8cB892d3DaA5F816939FC Implementation: 0xb5D23B66eb2208c897acea71eA8C5bc4ee026417	MATCH

Reservoir	Not deployed	N/A
InterestRateModel	Dependency	MATCH
JumpRateModel	0x05927DfA3f32FC2b0e1Ed680d9434858bCdEb926	MATCH
BaseJumpRateModelV2	Dependency	MATCH
JumpRateModelV2	0xB438F74c3cF937AB5d0060ba51043BeCcB013E78	MATCH
PriceOracle	Dependency	MATCH
AquariusBandPriceOracle	0xbA682AA7521eeBd4635f6E1380448E4422a57cD4	MATCH
AggregatorV2V3Interface	Dependency	MATCH
Ars	0xACaDD8eaC98F66a959E0DfaB66E3a548A6E57ce6	MATCH
GovernorBravoDelegate	0x21F39F9E2676D986F91fCd03E3e91C0c64dE54F6	MATCH
GovernorBravoDelegator	0xe2eD93B6C5741EABA44B93305c25c71423F365F3	MATCH
GovernorBravoInterfaces	Dependency	MATCH
PausingTimelock	0x0A415d557503dB29FD43343caE88135a5367EE17	MATCH
CarefulMath	Dependency	MATCH
EIP20Interface	Dependency	MATCH
EIP20NonStandardInterface	Dependency	MATCH
ErrorReporter	Dependency	MATCH
Exponential	Dependency	MATCH
ExponentialNoError	Dependency	MATCH
Maximillion	0x1B11B8AC1Acc2B2e8a45bBabd5616ce3d3519fD1	MATCH
Multicall	0xDcB2E4b4f0BD629964aA968B76AEf2Ce6f3C168B	MATCH
Multicall2	0x5d8D0e1225A2d8B410600fDB98e4A916370Efd7E	MATCH
SafeMath	Dependency	MATCH
AquariusLens	0xf854C6b57E6d8Ec3E790eC6Da1a13060d19549d4	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	6	4	2	-
Medium	7	1	-	6
Low	9	1	-	8
Informational	11	2	-	9
Total	**33**	**8**	**2**	**23**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 AErc20

No issues found.

1.3.2 AErc20Immutable

ID	Severity	Summary	Status
01	INFO	Inconsistency in naming convention	✓ RESOLVED

1.3.3 AToken

ID	Severity	Summary	Status
02	HIGH	Reentrancy possible across ATokens due to lack of adhering to checks-effects-interactions	PARTIAL
03	MEDIUM	Exchange rate of ATokens can be manipulated	ACKNOWLEDGED
04	LOW	Underlying token can be swept to admin if it has more than one address	✓ RESOLVED
05	INFO	`AToken` transfers do not revert but return a boolean	✓ RESOLVED

1.3.4 AErc20Delegator

ID	Severity	Summary	Status
06	HIGH	AToken implementation can be modified by admin	✓ RESOLVED

1.3.5 ATokenInterfaces

No issues found.

1.3.6 CEther

ID	Severity	Summary	Status
07	INFO	Inconsistency in naming conventions	ACKNOWLEDGED

1.3.7 Comptroller

ID	Severity	Summary	Status
08	HIGH	Supplied tokens that are not entered into markets can be seized through liquidation	✓ RESOLVED
09	MEDIUM	`fixBadAccruals` is an unnecessary function	✓ RESOLVED
10	MEDIUM	Loss of precision in `liquidateCalculateSeizeTokens` due to division before multiplication	ACKNOWLEDGED
11	LOW	Entering too many markets can result in out of gas in `getHypotheticalAccountLiquidityInternal`	ACKNOWLEDGED
12	LOW	`_grantArs` allows admin to grant arbitrary amount of ARS tokens to any address	ACKNOWLEDGED
13	INFO	`pauseGuardian` can unpause	ACKNOWLEDGED

1.3.8 ComptrollerInterface

No issues found.

1.3.9 ComptrollerStorage

ID	Severity	Summary	Status
14	INFO	`ComptrollerV7Storage` is unnecessary	ACKNOWLEDGED

1.3.10 Unitroller

ID	Severity	Summary	Status
15	HIGH	Unitroller implementation can be modified by admin	✓ RESOLVED
16	INFO	Unnecessary zero address check in `_acceptAdmin`	ACKNOWLEDGED

1.3.11 Reservoir

No issues found.

1.3.12 InterestRateModel

No issues found.

1.3.13 JumpRateModel

No issues found.

1.3.14 BaseJumpRateModelV2

ID	Severity	Summary	Status
17	INFO	`accrueInterest()` is not called before the interests rate model is manually adjusted by governance, causing the adjustment to work slightly in retrospect	ACKNOWLEDGED
18	INFO	Lack of validation within `updateJumpRateModelInternal`	ACKNOWLEDGED
19	INFO	`operatorGuardian` role cannot be transferred	ACKNOWLEDGED

1.3.15 JumpRateModelV2

No issues found.

1.3.16 PriceOracle

No issues found.

1.3.17 AquariusBandPriceOracle

ID	Severity	Summary	Status
20	HIGH	Price for any underlying asset can be set by admin	PARTIAL
21	HIGH	`getUnderlyingPrice` will have return a wrong price for underlying tokens which have more than 18 decimals due to integer underflow	✓ RESOLVED
22	MEDIUM	Lack of price freshness check results in usage of possibly stale prices	ACKNOWLEDGED
23	INFO	Single step `setAdmin` can result in loss of admin privileges due to human error	ACKNOWLEDGED

1.3.18 Ars

ID	Severity	Summary	Status
24	INFO	ARS tokens can be transferred to the ARS contract	ACKNOWLEDGED

1.3.19 GovernorBravoDelegate

ID	Severity	Summary	Status
25	MEDIUM	Possible passing of arbitrary proposals if ARS is cheap enough to acquire	ACKNOWLEDGED

1.3.20 GovernorBravoDelegator

No issues found.

1.3.21 GovernorBravoInterfaces

No issues found.

1.3.22 PausingTimelock

ID	Severity	Summary	Status
26	MEDIUM	Calling `renounceEmergencyAdmin` when a market in the Comptroller is paused can result in the timelock not being able to execute any further timelock transactions	ACKNOWLEDGED
27	MEDIUM	`renounceEmergencyAdmin` sets the wrong state variable to zero	ACKNOWLEDGED
28	LOW	admin can still call `cancelTransaction` even if any market's `borrow` and `mint` are paused	ACKNOWLEDGED
29	LOW	`pauseCount` and `isPaused` are unused state variables	ACKNOWLEDGED
30	LOW	Admin will be bypassed if any market in the Comptroller has `mint` and `borrow` paused	ACKNOWLEDGED

1.3.23 CarefulMath

No issues found.

1.3.24 EIP20Interface

No issues found.

1.3.25 EIP20NonStandardInterface

No issues found.

1.3.26 ErrorReporter

No issues found.

1.3.27 Exponential

No issues found.

1.3.28 ExponentialNoError

No issues found.

1.3.29 Maximillion

No issues found.

1.3.30 Multicall

ID	Severity	Summary	Status
31	LOW	Token approvals granted to the multicall contract can be misused to drain token balances by anyone	ACKNOWLEDGED

1.3.31 Multicall2

ID	Severity	Summary	Status
32	LOW	Token approvals granted to the multicall contract can be misused to drain token balances by anyone	ACKNOWLEDGED

1.3.32 SafeMath

No issues found.

1.3.33 AquariusLens

No issues found.

1.3.34 General Issues

ID	Severity	Summary	Status
33	LOW	Blocks instead of timestamp is used in the protocol	ACKNOWLEDGED

2 Findings

2.1 Lending and Core / AErc20

`AErc20` extends the `AToken` and handles the supply and borrows of ERC20 underlying tokens. If the underlying is the ARS token, the admin can delegate the ARS balance in this contract to an arbitrary delegatee.

2.1.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `_delegateArsLikeTo`

2.1.2 Issues & Recommendations

No issues found.

2.2 Lending and Core / AErc20Immutable

`AErc20Immutable` is the immutable, non-upgradable version of `AEerc20`, which does not use a proxy for storage. The Aquarius team has confirmed that this contract will not be used in production, and upgradable `AEerc20` tokens will be used instead.

2.2.1 Issues & Recommendations

Issue #01	Inconsistency in naming convention
Severity	INFORMATIONAL
Description	While the general naming convention of the Aquarius Protocol contracts follows the format of starting the capital A, `CDelegatorInterface` and `CDelegationStorage` are used in `AErc20Immutable`.
Recommendation	Consider renaming these variables to `ADelegatorInterface` and `ADelegationStorage`.
Resolution	RESOLVED

2.3 Lending and Core / AToken

Each `AToken` represents an asset that can be supplied and borrowed within the Aquarius Protocol. `AToken` contains the core logic of the supply and borrowing of a single underlying token in the Aquarius protocol. This is referred to as a "market" since it allows suppliers to transact by supplying the underlying token, using it as collateral, and borrowers to borrow it.

When users supply underlying tokens for lending, the `AToken` will be minted as a receipt token. The underlying tokens will be stored in the `AToken` contract, where others who have provided collateral are able to borrow the supplied tokens. Users who have exceeded the collateral ratio, defined as how much value they have borrowed over the value they have provided to the system, can be subjected to a liquidation of a portion of their position with a liquidation penalty. Timely liquidations are required to keep the protocol healthy with healthy debt (defined as properly collateralized debt).

Each token implementation has two other related contracts: A `Delegator` and `Delegate` contract. This is because ATokens are upgradeable, meaning that the token admin has the complete freedom to change their behavior over time. The `AErc20Delegator` is the proxy contract while the `AErc20Delegate` is the implementation contract that extends the `AErc20` contract.

2.3.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `initialize`

2.3.2 Issues & Recommendations

Issue #02	**Reentrancy possible across ATokens due to lack of adhering to checks-effects-interactions**
Severity	HIGH SEVERITY

Location	L694~
	`doTransferOut(redeemer, vars.redeemAmount);`
	
	`/* We write previously calculated values into storage */`
	`totalSupply = vars.totalSupplyNew;`
	`accountTokens[redeemer] = vars.accountTokensNew;`
	
	L786~
	`doTransferOut(borrower, borrowAmount);`
	
	`/* We write the previously calculated values into storage */`
	`accountBorrows[borrower].principal = vars.accountBorrowsNew;`
	`accountBorrows[borrower].interestIndex = borrowIndex;`
	`totalBorrows = vars.totalBorrowsNew;`

Description	Due to the lack of adhering to checks-effects-interactions pattern in `redeemFresh` and `borrowFresh`, underlying tokens are transferred out before the storage state is updated. While the `AToken` contract itself has reentrancy guards, there is no protocol level rentrancy guard, thus allowing tokens with callback functionality to allow reentrancy into another `AToken` before the state has been updated. This allows borrowing more than the account should be able to, as the state of the borrow state of the account has yet to be updated. This has been the cause of many exploits for Compound forks which have not rectified this issue and use underlyings with callback functionality such as ERC777 tokens.
Recommendation	Do the state changes first before the `doTransferOut` calls to adhere to checks-effects-interactions and prevent cross AToken reentrancy.

Resolution	PARTIALLY RESOLVED `doTransferOut` is now called after the state has been updated. Note that CEther is still using the old code as it cannot be upgraded, but as it is using `transfer` to send the native coin, reentrancy is prevented due to the gas limitations of transfer.

Issue #03	**Exchange rate of ATokens can be manipulated**
Severity	MEDIUM SEVERITY
Description	After an `AToken` has been deployed and added to the `Comptroller` as a market, a malicious actor would be able manipulate the exchange rate of the `AToken` with the following steps: 1. Mint `AToken` using a small amount of underlying to set the initial exchange rate 2. Redeem all but a small amount (e.g. 1 wei) of ATokens 3. Send a large amount of underlying tokens directly to the `AToken` contact. As there is only a total supply of 1 wei of `AToken` but a large underlying, the exchange rate would be skewed towards an extremely high rate. This is possible as the exchange rate used is based on the underlying token balance in the `AToken` contract. Any further mints would result in loss of funds for the depositor due to loss of precision when division is done by an extremely huge divisor. L535 `(vars.mathErr, vars.mintTokens) = divScalarByExpTruncate(vars.actualMintAmount, Exp({mantissa: vars.exchangeRateMantissa}));`

Recommendation	Consider minting a minimum liquidity amount to the burn address to make such exchange rate manipulation attacks more difficult. Also, the `mintTokens` amount can be checked to ensure that it is not zero. If no change is to be made to the code to ensure as little changes as possible from the original Compound codebase, ensure that when deploying and adding a new market to the `Comptroller`, the deployer is the first account to mint the AToken, and that no such exchange rate manipulation has taken place before opening usage of the `AToken` to users.
Resolution	ACKNOWLEDGED

Issue #04	**Underlying token can be swept to admin if it has more than one address**
Severity	LOW SEVERITY
Description	sweepToken can cause pricing issues of the AToken if the underlying token has more than 1 address (e.g. TUSD described in the blog below). https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2 By sweeping a token with a secondary address that has an entry point to the underlying token's contract functionality, it would be possible to circumvent the check that disallows the underlying tokens to be transferred out using the sweep function to the admin.
Recommendation	A recommended change would be to check the balance of the underlying before and after the transfer to ensure that the balance has not changed. Alternatively, a simpler but incomplete fix would be to only allow the admin to call this function. Additionally, underlying tokens should be vetted for such unexpected behavior before being added to the market.
Resolution	RESOLVED sweepToken can now only be called by admin.

Issue #05	AToken transfers do not revert but return a boolean
Severity	INFORMATIONAL
Description	Both `transfer` and `transferFrom` functions do not revert if there are any errors that prevent the transfer from successfully occuring, but return a `false` if the return value of `transferTokens` is non-zero.
Recommendation	Consider adding documentation to inform other protocols or contracts which do `AToken` transfers to check the return value of transfer related functions.
Resolution	RESOLVED Comments have been added to inform anyone using the transfer related functions to check for the return value.

2.4 Lending and Core / AErc20Delegator

AErc20Delegator is the proxy contract used for the AToken.

2.4.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `_setPendingImplementation`

2.4.2 Issues & Recommendations

Issue #06	AToken implementation can be modified by admin
Severity	HIGH SEVERITY
Description	The admin of the `Unitroller` can arbitrarily change the `Comptroller` contract being used as the implementation, and modify features which can cause sensitive state changes to the protocol.
Recommendation	Consider using a secure multi-signature solution as the admin address.
Resolution	RESOLVED The admin is confirmed to be a timelock contract.

2.5 Lending and Core / ATokenInterfaces

`ATokenInterfaces` is the interface for the `AToken` contract.

2.5.1 Issues & Recommendations

No issues found.

2.6 Lending and Core / CEther

`CEther` extends `AToken`, and handles the supply and borrows of native BTT. Unlike `AErc20` tokens, it is immutable and not upgradable.

2.6.1 Issues & Recommendations

Issue #07	Inconsistency in naming conventions
Severity	INFORMATIONAL
Description	In the codebase, while the naming convention for `ATokens` is to start with the capital A, and `CEther` does not follow the convention.
Recommendation	Consider renaming the contract to `AEther`.
Resolution	ACKNOWLEDGED

2.7 Lending and Core / Comptroller

The `Comptroller` is the core of the Aquarius protocol, containing the implementation which controls the markets. Operations against an `AToken` are required to pass verification against the `Comptroller` before it occurs, and an explicit validation after it occurred.

The `Comptroller` tracks the overall state of liquidity of each account and ensures that no actions can be taken that would leave a user undercollateralized, and that only liquidations can occur if the user is undercollateralized. In addition, the `Comptroller` also manages and distributes ARS liquidity mining rewards to suppliers and borrowers.

The protocol governance can tweak many essential variables within the `Comptroller` and thus all interactions with it should be carefully considered and validated by the community.

On a per market level, new markets can be added, and each of such markets can have supplying and borrowing paused, and have a cap set on borrow amounts. Additionally, reward speeds for supply and borrow of each market can be adjusted or stopped.

On a general level, transfers and liquidations of all `ATokens` can be paused or resumed.

Collateral quality is something important to be assessed before determining which tokens are used as collateral, the collateral factor, as well as the borrow cap to be set for each collateral. Some factors which can be used in this assessment include the price volatility of the asset, as well as the amount of on-chain liquidity. The former can lead to illiquid positions if a large amount of borrows are done against a collateral when there is a steep increase in price, followed by a steep decrease before sufficient liquidation can keep up. The latter can affect the liquidation of

large borrows of an asset as huge slippage can make liquidations unprofitable or even result in losses, thus discouraging seizing illiquid positions when they should be.

Borrow caps can be used as a method to cope with assets with such limitations. Do note that by default, the borrow cap for a newly added market is 0, which means that there is no cap enforced. The privileged roles will have to set the borrow cap for the market for it to come into effect. It is also highly recommended for the project team to take reference from existing risk management models such as those on https://gov.gauntlet.network/, but also be aware that even the same asset on different chains can have differing risks.

2.7.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `_setCloseFactor`
- `_setMarketBorrowCaps`
- `_setBorrowCapGuardian`
- `_setTeamPauseGuardian`
- `_renounceTeamPauseGuardian`
- `_renouncePauseGuardian`
- `_setPauseGuardian`
- `_setMintPaused`
- `_setBorrowPaused`
- `_setTransferPaused`
- `_setSeizePaused`
- `fixBadAccruals`
- `_grantArs`
- `_setArsSpeeds`
- `_setContributorArsSpeed`

- `_setPriceOracle`
- `_setCollateralFactor`
- `_setLiquidationIncentive`
- `_supportMarket`

The following function can be called by `borrowCapGuardian`:

- `_setMarketBorrowCaps`

The following functions can be called by `pauseGuardian`:

- `_setMintPaused`
- `_setBorrowPaused`
- `_setTransferPaused`
- `_setSeizePaused`

The following functions can be called by `teamPauseGuardian`:

- `_setPauseGuardian`
- `_renouncePauseGuardian`
- `_setTeamPauseGuardian`
- `_renounceTeamPauseGuardian`

2.7.1 Issues & Recommendations

Issue #08	**Supplied tokens that are not entered into markets can be seized through liquidation**
Severity	HIGH SEVERITY
Description	There is an issue where `cTokens` which are not entered in the market can be seized. Example: User has wBTC and USDC deposited into the protocol, but only entered the market for AwBTC. He borrows DAI. If wBTC price goes down and his position becomes illiquid, a liquidator can seize either wBTC (expected) or USDC (not expected). This is because there is a lack of checking if the collateral to be seized is entered in the market for an account.
Recommendation	Add checks in `liquidateBorrowAllowed` and `seizeAllowed` to ensure that the borrower has entered the market for the collateral to be seized. `require(markets[aTokenCollateral].accountMembership[borrower]);`
Resolution	RESOLVED The checks have been added.

Issue #09	`fixBadAccruals` is an unnecessary function
Severity	MEDIUM SEVERITY
Description	The above mentioned function was used in Compound Finance's Proposal 65 to correct over accrued reward tokens. https://compound.finance/governance/proposals/65 This bears no relation to the Aquarius Protocol and such an unnecessary function should be removed.
Recommendation	Remove the `fixBadAccruals` function.
Resolution	RESOLVED The function has been removed.

Issue #10	**Loss of precision in `liquidateCalculateSeizeTokens` due to division before multiplication**
Severity	MEDIUM SEVERITY

Location	L817~
	`numerator = mul_(Exp({mantissa: liquidationIncentiveMantissa}), Exp({mantissa: priceBorrowedMantissa}));`
	`denominator = mul_(Exp({mantissa: priceCollateralMantissa}), Exp({mantissa: exchangeRateMantissa}));`
	`ratio = div_(numerator, denominator);`
	
	`seizeTokens = mul_ScalarTruncate(ratio, actualRepayAmount);`

Description	In `liquidateCalculateSeizeTokens`, `seizeTokens` is calculated by doing division before multiplication. In either of the 3 cases below, which can cause the ratio to be zero if the numerator divided by the denominator is less than 1, causing liquidators to not receive any of the collateral during liquidation as seizeTokens would be zero: - `priceBorrowedMantissa` is very low - `priceCollateralMantissa` is very high - `exchangeRateMantissa` is very high This usually happens if the borrowed underlying asset has very few decimals and the collateral asset has many many decimals, causing the delta between them to be too large.

Recommendation	Consider doing multiplication first before division.
	`numerator = mul_(`
	`  mul_(`
	`  Exp({mantissa: liquidationIncentiveMantissa}),`
	`  Exp({mantissa: priceBorrowedMantissa})`
	`  ),`
	`  actualRepayAmount`
	`);`
	`denominator = mul_(`
	`  Exp({mantissa: priceCollateralMantissa}),`
	`  Exp({mantissa: exchangeRateMantissa})`
	`);`
	`ratio = div_(numerator, denominator);`
	
	`seizeTokens = truncate(ratio);`

Resolution	ACKNOWLEDGED

Issue #11	**Entering too many markets can result in out of gas in `getHypotheticalAccountLiquidityInternal`**
Severity	LOW SEVERITY
Description	Within `getHypotheticalAccountLiquidityInternal`, an account's liquidity in the protocol is calculated by looping over every asset in which the account has. If there are a large number of markets available, this could cause out of gas issues due to excessive number of loops done when calling `getHypotheticalAccountLiquidityInternal`, and could possibly result in denial of service for certain functionality, such as denying liquidations even when an account is illiquid.
Recommendation	Consider adding a cap to the number of assets an account can enter.
Resolution	ACKNOWLEDGED

Issue #12	**`_grantArs` allows admin to grant arbitrary amount of ARS tokens to any address**
Severity	LOW SEVERITY
Description	The admin has the capability to transfer out as much as the entire ARS balance of the Comptroller to any address.
Recommendation	Ensure that the admin account is secure to prevent draining rewards in the case of compromise, such as by using a secure multi-signature wallet as the admin.
Resolution	ACKNOWLEDGED

Issue #13	**pauseGuardian can unpause**
Severity	INFORMATIONAL
Description	Unlike the original implementation in Compound, which has the following line in the pause related functions to allow only the admin to unpause, this line has be removed in Aquarius' `Comptroller`. `require(msg.sender == admin || state == true, "only admin can unpause");` This allows the `pauseGuardian` to both pause and unpause functionality.
Recommendation	Confirm if this different behavior from Compound is intended.
Resolution	ACKNOWLEDGED

2.8 Lending and Core / ComptrollerInterface

`ComptrollerInterface` is the interface containing the interface for the `Comptroller`.

2.8.1 Issues & Recommendations

No issues found.

2.9 Lending and Core / ComptrollerStorage

`ComptrollerStorage` contains the definitions of state variables used by the `Comptroller`.

2.9.1 Issues & Recommendations

Issue #14	**`ComptrollerV7Storage` is unnecessary**
Severity	INFORMATIONAL
Description	The above mentioned contract was used in Compound Finance's Proposal 65 to correct over accrued reward tokens. https://compound.finance/governance/proposals/65 This bears no relation to the Aquarius Protocol and such unnecessary contact and storage should be removed.
Recommendation	Remove the `ComptrollerV7Storage` function.
Resolution	ACKNOWLEDGED

2.10 Lending and Core / Unitroller

`Unitroller` is the upgradeable proxy that delegates calls to the `Comptroller` for the execution logic. All states are stored in the `Unitroller`.

2.10.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `_setPendingImplementation`
- `_acceptImplementation`
- `_setPendingAdmin`
- `_acceptAdmin`
- `_become (in Comptroller)`

2.10.2 Issues & Recommendations

Issue #15	`Unitroller` implementation can be modified by admin
Severity	HIGH SEVERITY
Description	The admin of the `Unitroller` can arbitrarily change the `Comptroller` contract being used as the implementation, and modify features which can cause sensitive state changes to the protocol.
Recommendation	Consider using a secure multi-signature solution as the admin address.
Resolution	RESOLVED The admin is confirmed to be a timelock contract.

Issue #16	Unnecessary zero address check in `_acceptAdmin`
Severity	INFORMATIONAL
Description	In `_acceptAdmin`, `msg.sender == address(0)` conditional is unnecessary as `msg.sender` will not be `0x0`. `if (msg.sender != pendingAdmin || msg.sender == address(0)) {`
Recommendation	Consider removing the unnecessary zero address check.
Resolution	ACKNOWLEDGED

2.11 Lending and Core / Reservoir

The `Reservoir` contract will hold ARS tokens which are the rewards for the liquidity mining of the protocol. Over time, these rewards will be dripped into the `Comptroller` for distribution to suppliers and borrowers.

2.11.1 Issues & Recommendations

No issues found.

2.12 Interest Rate Models / InterestRateModel

Within Aquarius Loan, borrowers pay interest to suppliers. This interest rate is directly based on the utilization rate of the supply. If more of the supply is being lent out, a higher interest rate is paid. This is based on the economical principle of supply and demand to achieve a lending market in equilibrium. The degree to which the interest rate adjustment is specified in the interest rate models which should be carefully adjusted asset-by-asset. The interest rate models are responsible for calculating both the supply and borrow rate.

The `InterestRateModel` contract is the interface implemented by `JumpRateModel` and `JumpRateModelV2`.

2.12.1 Issues & Recommendations

No issues found.

2.13 Interest Rate Models / JumpRateModel

`JumpRateModel` is an interest model that depends on the utilization rate. It has a starting interest rate called `baseRatePerBlock` and goes up linearly with the utilization according to the `multiplierPerBlock` value. Once the kink utilization is reached, the rate can grow faster or slower according to `jumpMultiplierPerBlock`. The model thus has a base rate and two linear components.

blocksPerYear has been adjusted to 15512500, which tallies with the number of blocks at 2 seconds a block on the Bittorrent chain.

2.13.1 Issues & Recommendations

No issues found.

2.14 Interest Rate Models / BaseJumpRateModelV2

`BaseJumpRateModelV2` is the newer version of `JumpRateModel`, which allows the modification of parameters, which V1 does not allow.

`blocksPerYear` has been adjusted to `15512500`, which tallies with the number of blocks at 2 seconds a block on the Bittorrent chain.

2.14.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `updateJumpRateModel`
- `transferOwner`

The following functions can be called by `operatorGuardian`:

- `updateJumpRateModel`
- `transferOwner`

2.14.2 Issues & Recommendations

Issue #17	**`accrueInterest()` is not called before the interests rate model is manually adjusted by governance, causing the adjustment to work slightly in retrospect**
Severity	INFORMATIONAL
Description	The governance can manually adjust the slope and the two rates of the jump rate model through the `updateJumpRateModel` function. However, since no `accrueInterest()` is called on the related token before this, this might cause the new model to work slightly retroactively. This might cause stakers to unexpectedly pay a higher rate. This issue is marked as informational risk since it might not be trivial to add this requirement in since it would require either having the update go through the token or making the interest model aware of the token which might not be worth the trade-off.
Recommendation	Ensure that `accrueInterest` is called before any adjustment is made to the interest model.
Resolution	ACKNOWLEDGED

Issue #18	Lack of validation within `updateJumpRateModelInternal`
Severity	INFORMATIONAL
Description	The `updateJumpRateModelInternal` function does not validate that the parameters are within the expected ranges. Through user error, this might lead to undesirable effects.
Recommendation	Consider adding a reasonable upper/lower limit to validate against for the kink and multipliers.
Resolution	ACKNOWLEDGED

Issue #19	`operatorGuardian` role cannot be transferred
Severity	INFORMATIONAL
Description	Unlike the `transferOwner` function, which allows the transfer of the owner role to another address, there is no such functionality for the `operatorGuardian`. In the case the address set as the `operatorGuardian` is compromised, there is no way to transfer it.
Recommendation	Consider adding a similar function that allows the `operatorGuardian` to transfer the role to another address.
Resolution	ACKNOWLEDGED

2.15 Interest Rate Models / JumpRateModelV2

`JumpRateModelV2` is an interest model which implements `InterestRateModel` and `BaseJumpRateModelV2`.

2.15.2 Issues & Recommendations

No issues found.

2.16 Oracle / PriceOracle

`PriceOrcale` is extended by the `AquariusBandPriceOracle` to implement the logic of `getUnderlyingPrice` to return a scaled price based on the price of an underlying token reported by BAND protocol.

2.16.2 Issues & Recommendations

No issues found.

2.17 Oracle / AquariusBandPriceOracle

`AquariusBandPriceOracle` is the contract which the `Comptroller` uses as the oracle to obtain the price feeds for `AToken` underlying tokens.

There are 2 ways a price for an underlying token can be obtained: 1) manually setting by the admin, and 2) obtaining the rate from the BAND protocol oracle reference contract. The prices returned by this contract for each underlying token will determine the liquidity of each address' position, as well as whether an address can borrow, or be liquidated.

2.17.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `updateJumpRateModel`
- `transferOwner`

2.17.2 Issues & Recommendations

Issue #20	Price for any underlying asset can be set by admin
Severity	HIGH SEVERITY
Description	The `setUnderlyingPrice` and `setDirectPrice` functions allow the admin to set an arbitrary price for the underlying asset. In the case of compromise of the admin, or a malicious admin, prices can be manipulated and cause insolvency in the protocol. Human error when setting the prices (such as wrong scale of price) can also result in such issues.
Recommendation	Consider if such functions for the admin are necessary. If so, consider using a secure multi-signature solution as the admin address.
Resolution	ACKNOWLEDGED The client has indicated that the admin will be a multi-signature wallet. We will mark this as resolved once this has been done.
	PARTIALLY RESOLVED The admin of the contract is a Gnosis Safe multi-signature wallet, however the implementation of the contract is not verified and also not part of the audit scope.

Issue #21	**`getUnderlyingPrice` will have return a wrong price for underlying tokens which have more than 18 decimals due to integer underflow**
Severity	HIGH SEVERITY
Location	L58 `uint decimalDelta = 18-uint(token.decimals());`
Description	`getUnderlyingPrice` does not work with pricing of tokens which have more than 18 decimals. As the Solidity compiler used is below 0.8.0, there is no built in integer underflow on arithmetic by default, and this line will result in integer underflow if the token decimals is 19 or more. This will result in an extremely high `decimalDelta`, which when multiplied against the price returned by BAND protocol, will result in the wrong extremely high price.

Recommendation	With a constant of 18 as default decimals, decimal scaling can be done such as the following:
	
	`uint256 tokenDecimal = uint256(token.decimals());`
	`if(defaultDecimal == tokenDecimal) {`
	`  return price;`
	`} else if(defaultDecimal > tokenDecimal) {`
	`  return price.mul(10**(defaultDecimal.sub(tokenDecimal)));`
	`} else {`
	`  return price.div(10**(tokenDecimal.sub(defaultDecimal)));`
	`}`

Resolution	RESOLVED The suggested decimal scaling has been implemented.

Issue #22	Lack of price freshness check results in usage of possibly stale prices
Severity	MEDIUM SEVERITY
Description	Within `getUnderlyingPrice`, there is no freshness check to ensure that the price is not stale. https://docs.bandchain.org/band-standard-dataset/using-band-dataset/using-band-dataset-evm.html `getReferenceData` returns not only the rate, but also `lastUpdatedBase`, which is the last time the base price is updated. However, there is no check against a buffer to ensure that the price provided by BAND is updated in a reasonable timeframe. Similarly, this is the same for any manually set prices by the admin. This can result in the usage of stale prices which have not been updated in a long time, which can affect the solvency of the protocol as it can affect accounts' collateral health and liquidations.
Recommendation	There should be a check against a predetermined time buffer to ensure that the last update time of the price is not stale.
Resolution	ACKNOWLEDGED

Issue #23	**Single step `setAdmin` can result in loss of admin privileges due to human error**
Severity	INFORMATIONAL
Description	Currently, `setAdmin` is done in a single step by the current admin setting a new admin address. If there is any human error, such as using the wrong new admin address when calling `setAdmin`, access to admin privileges can be lost.
Recommendation	For `setAdmin`, consider doing a pull instead of push, where the current owner sets the new owner address and the new owner has to claim the ownership. This will prevent mis-setting of the owner to an invalid address, resulting in loss of ownership functionality of the contract.
Resolution	ACKNOWLEDGED

2.18 Governance & Timelocks / Ars

Ars is the native token of the Aquarius Protocol. It is used for the governance of the protocol, and can be used to vote for proposals in `GovernorBravo` based on delegation power. Accounts can delegate their votes to another delegatee address.

There is a total fixed supply of 1 billion ARS tokens, all minted to the specified account upon the deployment of the token contract.

2.18.1 Token Overview

Address	TBD
Token Supply	1,000,000,000 (1 billion)
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	1,000,000,000 (1 billion)

2.18.1 Issues & Recommendations

Issue #24	ARS tokens can be transferred to the ARS contract
Severity	INFORMATIONAL
Description	In the current implementation of the ARS token contract, there is no prevention of sending ARS tokens to the token contract in the `transfer` related function. This is a common error users make when transferring tokens, and can result in unnecessary loss of user funds.
Recommendation	Prevent transfers of Ars to the Ars token contract as there is no use case for it. Doing so can prevent loss of user funds due to users mistaking the token contract address as the to address when sending. Within `_transferTokens`: `require(to != address(this));`
Resolution	ACKNOWLEDGED

2.19 Governance & Timelocks / GovernorBravoDelegate

`GovernorBravoDelegate` contains the implementation logic used by `GovernorBravoDelegator`. It will act as the Governance contract which is the owner of `PausingTimelock`. Any timelock transactions to be queued, executed or canceled will have to be proposed and voted on using this governance contract.

There are a 4 steps for a proposal:

- **Step 1: Propose.** Anyone can propose a maximum of 10 actions, provided the account doing the proposal exceeds the proposal threshold, or is a whitelisted account.

- **Step 2: Vote.** During the voting period, which is the start until the end of the proposal, anyone can vote with the voting power they have at the start block of the proposal. Votes can either be against, for or to abstain.

- **Step 3: Queue.** After the end block of a proposal, if there are more `forVotes` than `againstVotes`, and the `forVotes` are more than or equal to the `quorumVotes`, it will be considered as successful, and can be queued for execution after the timelock delay has passed.

- **Step 4: Execute.** Once the timelock delay has passed, a queued proposal can be executed.

At any point of time before a proposal is executed, it can be canceled under the following situations:

- If the proposer is not a whitelisted account, the account attempting to cancel the proposal must have more than or equal to the `proposalThreshold` in voting power in 1 block before the cancellation is done.

- If the proposer is a whitelisted account, the canceller has to be the `whitelistGuardian`, and it must also have more than or equal to the `proposalThreshold` in voting power in 1 block before the cancellation is done.

2.19.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `_setVotingDelay`
- `_setVotingPeriod`
- `_setProposalThreshold`
- `_setWhitelistAccountExpiration`
- `_setWhitelistGuardian`
- `_initiate`
- `_setPendingAdmin`

The following function can be called by the whitelistGuardian:

- `_setWhitelistAccountExpiration`

2.19.2 Issues & Recommendations

Issue #25	**Possible passing of arbitrary proposals if ARS is cheap enough to acquire**
Severity	MEDIUM SEVERITY
Description	Depending on the amount initialized, the requirements of proposing a proposal can lie within the range of 5,000,000 and 10,000,000 ARS, and quorum vote requires 20,000,000 ARS. If the financial benefits of being to be able to execute (e.g. modify collateral ratio) a proposal exceeds the cost of obtaining the amount required to propose and surpass quorum vote, an attacker might do so. At this current point of time, it is unclear what the economic value of ARS is.
Recommendation	Constantly monitor proposals as they are passed. If a proposal is malicious or can have negative effects towards the protocol and its users, it should immediately be canceled by the whitelist guardian.
Resolution	ACKNOWLEDGED

2.20 Governance & Timelocks / GovernorBravoDelegator

`GovernorBravoDelegator` is the proxy which uses `GovernorBravoDelegate` as the implementation.

2.20.1 Issues & Recommendations

No issues found.

2.21 Governance & Timelocks / GovernorBravoInterfaces

`GovernorBravoInterfaces` is the interface for `GovernorBravoDelegate`.

2.21.1 Issues & Recommendations

No issues found.

2.22 Governance & Timelocks / PausingTimelock

`PausingTimelock` is a fork of the Timelock contract by Compound, which a change made to allow the `queueTransaction` and `executeTransaction` to be called by an `emergencyAdmin` role, instead of the admin role (which should be the governance contract), if there is one or more markets with mint and borrow paused on the Comptroller.

`PausingTimelock` is to be the admin roles related to the `Comptroller`, `Unitroller`, and `ATokens`.

There is a minimum delay of 2 days, a maximum delay of 30 days, and a grace period of 14 days for transactions ready for execution to be executed.

2.22.1 Priviledged Operations

The following functions can be called by the admin of the contract:

- `queueTransaction`
- `cancelTransaction`
- `executeTransaction`
- `renounceEmergencyAdmin`

The following functions can be called by the emergencyAdmin:

- `queueTransaction` (only if one or more markets are paused in comptroller)
- `executeTransaction` (only if one or more markets are paused in comptroller)
- `renounceEmergencyAdmin`

The following functions can be called by the Timelock itself:

- `setDelay`
- `setPendingAdmin`
- `setPendingEmergencyAdmin`
- `setComptroller`

2.22.2 Issues & Recommendations

Issue #26	**Calling `renounceEmergencyAdmin` when a market in the Comptroller is paused can result in the timelock not being able to execute any further timelock transactions**
Severity	MEDIUM SEVERITY
Description	If there are one or more markets paused in the Comptroller, all the function calls now have to be executed by `emergencyAdmin`. In such a state, if `renounceEmergencyAdmin` is called to set the `emergencyAdmin` address to the zero address, the timelock cannot execute any transactions any more, including `setPendingEmergencyAdmin`. This makes the timelock unable to function.
Recommendation	Consider removing the `renounceEmergencyAdmin` function.
Resolution	ACKNOWLEDGED

Issue #27	**renounceEmergencyAdmin sets the wrong state variable to zero**
Severity	MEDIUM SEVERITY

Location	L164~
	`function renounceEmergencyAdmin() public {`
	`    require(msg.sender == admin || msg.sender == emergencyAdmin, "Timelock:: call must come from admin or emergency admin");`
	
	`    pendingAdmin = address(0);`
	
	`    emit RenounceEmergencyAdmin();`
	`}`

Description	Calling `renounceEmergencyAdmin` sets `pendingAdmin` instead of `emergencyAdmin` to the zero address. This is not doing what the function name suggests the behavior should be.
Recommendation	`renounceEmergencyAdmin` should set `emergencyAdmin` to the zero address to correctly reflect the behavior of the function.
Resolution	ACKNOWLEDGED

Issue #28	**admin can still call `cancelTransaction` even if any market's `borrow` and `mint` are paused**
Severity	LOW SEVERITY
Description	Unlike `queueTransaction` and `executeTransaction`, `cancelTransaction` does not have the same if/else statement to check if `msg.sender` is `emergencyAdmin` if there is a market paused. This results in inconsistency in the timelock functionality, and allows the admin to cancel transactions instead of the `emergencyAdmin`, even if there is a market paused in the Comptroller.
Recommendation	Consider adding the same if/else statement in `cancelTransaction`.
Resolution	ACKNOWLEDGED

Issue #29	**`pauseCount` and `isPaused` are unused state variables**
Severity	LOW SEVERITY
Description	`pauseCount` is a state variable which is undefined, so it will be zero. It is used as part of the `txHash` calculation in `queueTransaction`, `executeTransaction`, and passed as a parameter in `cancelTransaction`. The state variable is never incremented, meaning that all transactions queued will have a `pauseCount` of zero. Also, `isPaused` is never modified after being declared.
Recommendation	Consider what the purpose of the `pauseCount` is, and remove it from the calculation of the `txHash` if unnecessary. `isPaused` can also be removed if not used.
Resolution	ACKNOWLEDGED

Issue #30	**Admin will be bypassed if any market in the Comptroller has mint and borrow paused**
Severity	LOW SEVERITY
Description	As long as there is at least a single market that has both borrow and mint paused, the timelock's queue and execution will no longer be allowed for the `admin`, but the `emergencyAdmin`. This can be done by the `pauseGuardian` calling `_setMintPaused` and `_setBorrowPaused` for any market in the Comptroller. In the case where the original admin is some sort of governance functionality related address, this can allow the bypass of such governance functionality.
Recommendation	Confirm the need for the `emergencyAdmin`, and remove it if unnecessary.
Resolution	ACKNOWLEDGED

2.23 Libraries & Peripheral / CarefulMath

As a compiler version below Solidity 0.8.0 is used, there is no in-built integer overflow/underflow protection for native arithmetic.

Derived from OpenZeppelin's `SafeMath` library, `CarefulMath` guards against integer overflow/underflow for addition, subtraction, multiplication and division. However, it does not revert if overflow/underflow is detected, but returns a `MathError` as the first return parameter.

It should be noted that if `CarefulMath` functions are used for calculations, it is necessary to verify that the `MathError` returned is zero. Non-zero values would indicate that there is an integer overflow/underflow, and should be handled accordingly.

2.23.1 Issues & Recommendations

No issues found.

2.24 Libraries & Peripheral / EIP20Interface

`EIP20Interface` is the interface for `EIP20` contracts.

2.24.1 Issues & Recommendations

No issues found.

2.25 Libraries & Peripheral / EIP20NonStandardInterface

`EIP20Interface` is the interface for non standard `EIP20` contracts.

2.25.1 Issues & Recommendations

No issues found.

2.26 Libraries & Peripheral / ErrorReporter

`ErrorReporter` contains the error codes that are used in the money market protocol.

2.26.1 Issues & Recommendations

No issues found.

2.27 Libraries & Peripheral / Exponential

`Exponential` is a module for storing fixed-precision decimals. For each exponential function, the first return parameter is an error code which should be validated to not be zero to ensure that there are no errors.

2.27.1 Issues & Recommendations

No issues found.

2.28 Libraries & Peripheral / ExponentialNoError

Similar to `Exponential`, `ExponentialNoError` is a module for storing fixed-precision decimals, but contains arithmetic functions that do not return an error, but instead revert on error.

2.28.1 Issues & Recommendations

No issues found.

2.29 Libraries & Peripheral / Maximillion

`Maximillion` is used as a peripheral contract to facilitate the repaying on behalf for borrows of `CEther`. It is used to pay an excess of `CEther` borrows, and refund the remaining amount to the caller of the function.

This allows the full repayment of any `CEther` borrows in a single function call without the need to worry about interest accumulation between the time the transaction is broadcast and actually executed on chain.

2.29.1 Issues & Recommendations

No issues found.

2.30 Libraries & Peripheral / Multicall

Mulitcall is a peripheral contract which allows calling multiple state modifying calls using the aggregate function. All calls have to succeed, and if any single call fails, the whole function will revert.

2.30.1 Issues & Recommendations

Issue #31	**Token approvals granted to the multicall contract can be misused to drain token balances by anyone**
Severity	LOW SEVERITY
Description	While it is unclear what the purpose of the multicall contract is, if any token allowance is to be granted to the multicall contract, it will be possible to use the allowance granted by a user to the multicall and transfer the user's token balance away to an arbitrary address by calling the token contract's `transferFrom` function.
Recommendation	Confirm what the purpose of the multicall is. Ensure that token allowances are never granted to this contract.
Resolution	ACKNOWLEDGED

2.31 Libraries & Peripheral / Multicall2

Multicall2 is similar to Multicall, with a new tryAggregate function which allows a batch of calls to successfully complete even if there are 1 or more calls that revert.

2.31.1 Issues & Recommendations

Issue #32	**Token approvals granted to the multicall contract can be misused to drain token balances by anyone**
Severity	LOW SEVERITY
Description	While it is unclear what the purpose of the multicall contract is, if any token allowance is to be granted to the multicall contract, it will be possible to use the allowance granted by a user to the multicall and transfer the user's token balance away to an arbitrary address by calling the token contract's `transferFrom` function.
Recommendation	Confirm what the purpose of the multicall is. Ensure that token allowances are never granted to this contract.
Resolution	ACKNOWLEDGED

2.32 Libraries & Peripheral / SafeMath

As a compiler version below Solidity 0.8.0 is used, there is no in-built integer overflow/underflow protection for native arithmetic. Derived from OpenZeppelin's SafeMath library, `SafeMath` guards against integer overflow/underflow for addition, subtraction, multiplication and division.

Unlike `CarefulMath`, `SafeMath` reverts on detection of integer overflow/underflow.

2.32.1 Issues & Recommendations

No issues found.

2.33 Libraries & Peripheral / AquariusLens

AquariusLens is used as a peripheral contract to batch view information about the Comptroller, ATokens and the Governance of the protocol.

2.33.1 Issues & Recommendations

No issues found.

2.34 General Issues

In this section, we list general issues that are applied to the protocol as a whole.

Issue #33	Blocks instead of timestamp is used in the protocol
Severity	LOW SEVERITY
Description	Blocks instead of timestamp are used for the calculation of interest and rewards. While Bittorrent Chain has a constant block time of 2 seconds, and this should not affect the consistency in a protocol that uses block numbers, it is something to take into consideration, especially since there could be an issue if blocks are not produced in a timely manner or the chain goes down. In the case of any chain down time, or irregularities in block time, interest calculation as well as reward calculation will be affected.
Recommendation	Considering switching all instances of block usages in the protocol to timestamp.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY