EtherAuthority

FORTRESS PROTOCOL SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer:	JetFuel Team (https://jetfuel.finance)
Prepared on:	19/03/2021
Platform:	Binance Smart Chain
Language:	Solidity
Audit Type:	Extensive

audit@etherauthority.io

Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Documents

Name	Smart Contract Code Review and Security Analysis Report for FORTRESS
Platform	Binance Smart Chain / Solidity
File 1	FTS.sol
File 1 MD5 hash	16184E612400DCE0013F54FB60212FF2
File 1 Testnet Contract URL	https://testnet.bscscan.com/address/0xd8db4d409bc9461d3395574d9596e59ded1fba5e#code
File 2	FAI.sol
File 2 MD5 hash	D09CB24C6EA078EEC2F38348244E215C
File 2 Testnet Contract URL	https://testnet.bscscan.com/address/0x3B13b1af99b6d75D532C4E234fb2c3aE62744b73#code
File 3	Unitroller.sol
File 3 MD5 hash	2CA395D65CCA9872B141A39761850117
File 3 Testnet Contract URL	https://testnet.bscscan.com/address/0xB24DaCE5343A97fc0E82584Ecd52c7e54ABBda09#code
File 4	Comptroller.sol
File 4 MD5 hash	B9215D650D78D056DBF26DFB949FF9DF
File 4 Testnet Contract URL	https://testnet.bscscan.com/address/0x0996754B35B71d0F5A43d372Ef79cF5a4e852208#code
File 5	FAIUnitroller.sol
File 5 MD5 hash	D363232ED72C9F0B78DE8AB0140EF5D3
File 5 Testnet Contract URL	https://testnet.bscscan.com/address/0x1221622cC891Ee3f8A0F779D51f5fe72AF53c290#code
File 6	FAIController.sol
File 6 MD5 hash	28668F1BCD8DD4BFA514789790842396

File 6 Testnet Contract URL	https://testnet.bscscan.com/address/0x2a81348D13cd4Dc5A886C1Fb6CB4115C83767f09#code
File 7	SFTVaultProxy.sol
File 7 MD5 hash	5B2F3BA1C4777003C7BF7AE3D1914043
File 7 Testnet Contract URL	https://testnet.bscscan.com/address/0xB61B6a4f486B273A25AAa263D50d8e6b91B78eb4#code
File 8	SFTVault.sol
File 8 MD5 hash	538ABD88AFDC40BC1D1487216BCB5F58
File 8 Testnet Contract URL	https://testnet.bscscan.com/address/0xd65C9f4e37a6dB0f6F0ABEc584997D8ac110bE79#code
File 9	FortressLens.sol
File 9 MD5 hash	CDEA199BB76B31007252700B9469371C
File 9 Testnet Contract URL	https://testnet.bscscan.com/address/0xb24dace5343a97fc0e82584ecd52c7e54abbda09#code
File 10	WhitePaperInterestRateModel.sol
File 10 MD5 hash	111F06FACC068AC86133F754F4396F40
File 10 Testnet Contract URL	https://testnet.bscscan.com/address/0x3B13b1af99b6d75D532C4E234fb2c3aE62744b73#code
File 11	FortressPriceOracle.sol
File 11 MD5 hash	4E39ACF0B27511860B168F6C76C85B09
File 11 Testnet Contract URL	https://testnet.bscscan.com/address/0xb24dace5343a97fc0e82584ecd52c7e54abbda09#code
File 12	FBep20Delegate.sol
File 12 MD5 hash	F48021DCC4AF0D5AFA2B9712C61484E7
File 12 Testnet Contract URL	https://testnet.bscscan.com/address/0x0AE91e9BbCEf6d616760bFEbDa821099C531E61a#code
File 13	FBep20Delegator.sol (for fUSDC)
File 13 MD5 hash	599302613299E41D2043DF3459D858C

File 13 Testnet Contract URL	https://testnet.bscscan.com/address/0xF984655CB544Cc25deE85A4d6b1374410F9672c7#code
File 14	Timelock.sol
File 14 MD5 hash	8025863D3B4036F7FFC40E53CAD717AB
File 14 Testnet Contract URL	https://testnet.bscscan.com/address/0x1CEBf172C2c67e25292ED76Af534687bA9d86FcB#code
File 15	GovernorAlpha.sol
File 15 MD5 hash	F6CFEA696869B67C6414B4616EA0A6F6
File 15 Testnet Contract URL	https://testnet.bscscan.com/address/0x5589DD6f2FBFE7C668D986Dce031fEF2A848Ca31#code

Introduction

We were contracted by the JetFuel team to perform the Security audit of the smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on 19/03/2021.

Audit type was Extensive Audit. Which means this audit is concluded based on Extensive audit scope. This document outlines all the findings as well as AS-IS overview of the smart contract codes.

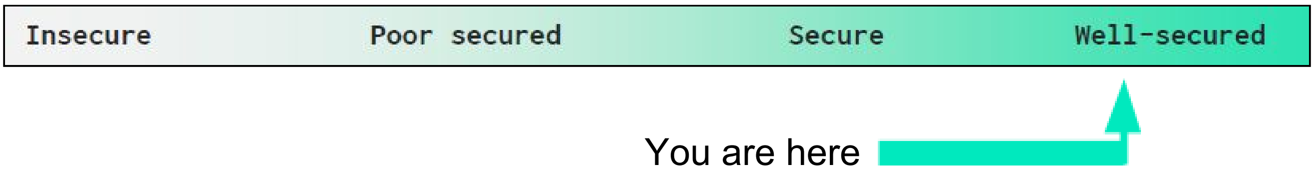
Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	Assert() misuse	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	The maximum limit for mintage not set	Moderted
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Executive Summary

According to the **extensive** audit assessment, Customer`s solidity smart contract is **well secured**.



We used various tools like SmartDec, Mythril, Slither and Remix IDE. At the same time this finding is based on critical analysis of the menual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all found issues can be found in the Audit overview section.

We found 0 high, 1 medium and 1 low and some very low level issues.

Code Quality

Fortress protocol consists of 14 core smart contract files. These smart contracts also contain Libraries, Smart contract inherits and Interfaces. These are compact and well written contracts.

The libraries in the Fortress protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Fortress protocol.

The Fortress team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Overall, code parts are well commented. Commenting can provide rich documentation for functions, return variables and more. Ethereum Natural Language Specification Format (NatSpec) is used, which is a good thing.

Documentation

We were given Fortress smart contracts in the form of a Bscscan testnet website links. The hashes of those files and their links are mentioned above in the table.

As mentioned above, most code parts are well commented. so anyone can quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol. It also provided a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, Fortress smart contracts depend on external smart contracts data, which is provided by oracle smart contract.

AS-IS overview

Fortress.sol contract overview

Fortress protocol is a decentralized marketplace for Lenders and Borrowers with Borderless Stablecoins. Following are the main components of core smart contracts.

FTS.sol

(1) Inherited contracts

(a) Owned: ownership contract
(b) Tokenlock: Token locking contract controlled by owner

(2) Events

(a) event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);
(b) event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);
(c) event Transfer(address indexed from, address indexed to, uint256 amount);
(d) event Approval(address indexed owner, address indexed spender, uint256 amount);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	allowance	read	Passed	No Issue	Passed
3	approve	write	Passed	No Issue	Passed
4	balanceOf	read	Passed	No Issue	Passed
5	transfer	write	Passed	No Issue	Passed
6	transferFrom	write	Passed	No Issue	Passed
7	delegate	write	Passed	No Issue	Passed
8	delegateBySig	write	Passed	No Issue	Passed
9	getCurrentVotes	read	Passed	No Issue	Passed

10	getPriorVotes	read	Infinite loop possibility	Votes must not be excessive	Passed
11	_delegate	internal	Passed	No Issue	Passed
12	_transferTokens	internal	Passed	No Issue	Passed
13	_moveDelegates	internal	Passed	No Issue	Passed
14	_writeCheckpoint	internal	Passed	No Issue	Passed
15	safe32	read	Passed	No Issue	Passed
16	safe96	read	Passed	No Issue	Passed
17	add96	read	Passed	No Issue	Passed
18	sub96	read	Passed	No Issue	Passed
19	getChainId	read	Passed	No Issue	Passed

FAI.sol

(1) Inherited contracts

(a)LibNote: provides log event

(2) Events

(a) event Approval(address indexed src, address indexed guy, uint wad);

(b) event Transfer(address indexed src, address indexed dst, uint wad);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	rely	write	Passed	No Issue	Passed
2	deny	write	Passed	No Issue	Passed
3	add	read	Passed	No Issue	Passed
4	sub	read	Passed	No Issue	Passed
5	constructor	write	Passed	No Issue	Passed
6	transfer	write	Passed	No Issue	Passed
7	transferFrom	write	Passed	No Issue	Passed
8	mint	write	No max minting	Unitroller regulates minting	Passed with consent
9	burn	write	Passed	No Issue	Passed
10	approve	write	Passed	No Issue	Passed
11	push	write	Passed	No Issue	Passed
12	pull	write	Passed	No Issue	Passed

13	move	write	Passed	No Issue	Passed
14	permit	write	Passed	No Issue	Passed

Unitroller.sol

(1) Inherited contracts

(a) UnitrollerAdminStorage: Admin contract for unitroller
(b) ComptrollerErrorReporter: Error reporting contract

(2) Events

(a) event NewPendingImplementation(address oldPendingImplementation, address newPendingImplementation);
(b) event NewImplementation(address oldImplementation, address newImplementation);
(c) event NewPendingAdmin(address oldPendingAdmin, address newPendingAdmin);
(d) event NewAdmin(address oldAdmin, address newAdmin);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	_setPendingImplementation	write	Passed	No Issue	Passed
3	_acceptImplementation	write	Passed	No Issue	Passed
4	_setPendingAdmin	write	Passed	No Issue	Passed
5	_acceptAdmin	write	Passed	No Issue	Passed
6	fallback	write	Delegates to implementation	No Issue	Passed

Comptroller.sol

(1) Inherited contracts

(a) ComptrollerV3Storage: Storage variables of comptroller
(b) ComptrollerInterface: Interfaces for comptroller
(c) ComptrollerErrorReporter: Error reporter contract
(d) Exponential: Library for exponential math functions

(2) Events

(a) event MarketListed(FToken fToken);
(b) event MarketEntered(FToken fToken, address account);
(c) event MarketExited(FToken fToken, address account);
(d) event NewCloseFactor(uint oldCloseFactorMantissa, uint newCloseFactorMantissa);
(e) event NewCollateralFactor(FToken fToken, uint oldCollateralFactorMantissa, uint newCollateralFactorMantissa);
(f) event NewLiquidationIncentive(uint oldLiquidationIncentiveMantissa, uint newLiquidationIncentiveMantissa);
(g) event NewMaxAssets(uint oldMaxAssets, uint newMaxAssets);
(h) event NewPriceOracle(PriceOracle oldPriceOracle, PriceOracle newPriceOracle);
(i) event NewFAIVaultInfo(address vault_, uint releaseStartBlock_, uint releaseInterval_);
(j) event NewPauseGuardian(address oldPauseGuardian, address newPauseGuardian);
(k) event ActionPaused(string action, bool pauseState);
(l) event ActionPaused(FToken fToken, string action, bool pauseState);
(m) event MarketFortress(FToken fToken, bool isFortress);
(n) event NewFortressRate(uint oldFortressRate, uint newFortressRate);
(o) event NewFortressFAIRate(uint oldFortressFAIRate, uint newFortressFAIRate);
(p) event NewFortressFAIVaultRate(uint oldFortressFAIVaultRate, uint newFortressFAIVaultRate);
(q) event FortressSpeedUpdated(FToken indexed fToken, uint newSpeed);
(r) event DistributedSupplierFortress(FToken indexed fToken, address indexed supplier, uint fortressDelta, uint fortressSupplyIndex);

(s) event DistributedBorrowerFortress(FToken indexed fToken, address indexed borrower, uint fortressDelta, uint fortressBorrowIndex);

(t) event DistributedFAIMinterFortress(address indexed faiMinter, uint fortressDelta, uint fortressFAIMintIndex);

(u) event DistributedFAIVaultFortress(uint amount);

(v) event NewFAIController(FAIControllerInterface oldFAIController, FAIControllerInterface newFAIController);

(w) event NewFAIMintRate(uint oldFAIMintRate, uint newFAIMintRate);

(x) event ActionProtocolPaused(bool state);

(3) Functions

Sl	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	getAssetsIn	read	Passed	No Issue	Passed
3	checkMembership	read	Passed	No Issue	Passed
4	enterMarkets	write	Infinite loop possibility	Keep fTokens limited	Passed
5	addToMarketInternal	internal	Passed	No Issue	Passed
6	exitMarket	write	Passed	No Issue	Passed
7	mintAllowed	write	Passed	No Issue	Passed
8	mintVerify	write	Passed	No Issue	Passed
9	redeemAllowed	write	Passed	No Issue	Passed
10	redeemAllowedInternal	internal	Passed	No Issue	Passed
11	redeemVerify	write	Passed	No Issue	Passed
12	borrowAllowed	write	Passed	No Issue	Passed
13	borrowVerify	write	Passed	No Issue	Passed
14	repayBorrowAllowed	write	Passed	No Issue	Passed
15	repayBorrowVerify	write	Passed	No Issue	Passed
16	liquidateBorrowAllowed	write	Passed	No Issue	Passed
17	liquidateBorrowVerify	write	Passed	No Issue	Passed
18	seizeAllowed	write	Passed	No Issue	Passed
19	seizeVerify	write	Passed	No Issue	Passed
20	transferAllowed	write	Passed	No Issue	Passed
21	transferVerify	write	Passed	No Issue	Passed
22	getAccountLiquidity	read	Passed	No Issue	Passed

23	getHypotheticalAccountLiquidity	read	Passed	No Issue	Passed
24	getHypotheticalAccountLiquidityInternal	internal	Infinite loop possibility	Keep assets limited	Passed
25	liquidateCalculateSeizeTokens	read	Passed	No Issue	Passed
26	_setPriceOracle	write	Passed	No Issue	Passed
27	_setCloseFactor	write	Passed	No Issue	Passed
28	_setCollateralFactor	write	Passed	No Issue	Passed
29	_setMaxAssets	write	Passed	No Issue	Passed
30	_setLiquidationIncentive	write	Passed	No Issue	Passed
31	_supportMarket	write	Passed	No Issue	Passed
32	_addMarketInternal	internal	Passed	No Issue	Passed
33	_setProtocolPaused	write	Passed	No Issue	Passed
34	_setFAIController	write	Passed	No Issue	Passed
35	_setFAIMintRate	write	Passed	No Issue	Passed
36	_setTreasuryData	write	Passed	No Issue	Passed
37	_become	write	Passed	No Issue	Passed
38	refreshFortressSpeeds	write	Passed	No Issue	Passed
39	refreshFortressSpeedsInternal	internal	Infinite loop possibility	Markets must be limited	Passed
40	updateFortressSupplyIndex	internal	Passed	No Issue	Passed
41	updateFortressBorrowIndex	internal	Passed	No Issue	Passed
42	distributeSupplierFortress	internal	Passed	No Issue	Passed
43	distributeBorrowerFortress	internal	Passed	No Issue	Passed
44	distributeFAIMinterFortress	write	Passed	No Issue	Passed
45	transferFTS	internal	Passed	No Issue	Passed
46	claimFortress	write	Infinite loop possibility	Array length must be limited	Passed
47	_setFortressRate	write	Passed	No Issue	Passed
48	_setFortressFAIRate	write	Passed	No Issue	Passed
49	_setFortressFAIVaultRate	write	Passed	No Issue	Passed

50	_setFAIVaultInfo	write	Passed	No Issue	Passed
51	_addFortressMarkets	write	Infinite loop possibility	Array length must be limited	Passed
52	_addFortressMarketInternal	internal	Passed	No Issue	Passed
53	_dropFortressMarket	write	Passed	No Issue	Passed
54	getAllMarkets	read	Passed	No Issue	Passed
55	getBlockNumber	read	Passed	No Issue	Passed
56	getFTSAddress	read	hard coded address	Keep it in a variable	Passed
57	setMintedFAIOf	write	Passed	No Issue	Passed
58	releaseToVault	write	Passed	No Issue	Passed

FAIUnitroller.sol

(1) Inherited contracts

(a) FAIUnitrollerAdminStorage: Admin contract for FAI unitroller

(b)FAIControllerErrorReporter: Error reporting contract

(2) Events

(a) event NewPendingImplementation(address oldPendingImplementation, address newPendingImplementation);

(b) event NewImplementation(address oldImplementation, address newImplementation);

(c) event NewPendingAdmin(address oldPendingAdmin, address newPendingAdmin);

(d) event NewAdmin(address oldAdmin, address newAdmin);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	_setPendingImplementation	write	Passed	No Issue	Passed
3	_acceptImplementation	write	Passed	No Issue	Passed
4	_setPendingAdmin	write	Passed	No Issue	Passed

5	_acceptAdmin	write	Passed	No Issue	Passed
6	fallback	write	Delegates to implementation	No Issue	Passed

FAIController.sol

(1) Inherited contracts

(a) FAIControllerStorage: Admin contract for FAI unitroller
(b) FAIControllerErrorReporter: Error reporting contract
(c) Exponential: math functions for exponential

(2) Events

(a) event NewComptroller(ComptrollerInterface oldComptroller, ComptrollerInterface newComptroller);
(b) event MintFAI(address minter, uint mintFAIAmount);
(c) event RepayFAI(address repayer, uint repayFAIAmount);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	mintFAI	write	Passed	No Issue	Passed
2	repayFAI	write	Passed	No Issue	Passed
3	_initializeFortressFAIState	write	Passed	No Issue	Passed
4	updateFortressFAIMintIndex	write	Passed	No Issue	Passed
5	calcDistributeFAIMinterFortress	write	Passed	No Issue	Passed
6	_setComptroller	write	Passed	No Issue	Passed
7	_become	write	Passed	No Issue	Passed
8	getMintableFAI	write	Infinite loop possibility	Keep array length limited	Passed
9	getBlockNumber	read	Passed	No Issue	Passed
10	getFAIAddress	read	hard coded address	Keep it in a variable	Passed

SFTVaultProxy.sol

(1) Inherited contracts

(a) SFTVaultAdminStorage: Admin contract for FAI unitroller

(b) SFTVaultErrorReporter: Error reporting contract

(2) Events

(a) event NewPendingImplementation(address oldPendingImplementation, address newPendingImplementation);

(b) event NewImplementation(address oldImplementation, address newImplementation);

(c) event NewPendingAdmin(address oldPendingAdmin, address newPendingAdmin);

(d) event NewAdmin(address oldAdmin, address newAdmin);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	_setPendingImplementation	write	Passed	No Issue	Passed
3	_acceptImplementation	write	Passed	No Issue	Passed
4	_setPendingAdmin	write	Passed	No Issue	Passed
5	_acceptAdmin	write	Passed	No Issue	Passed
6	fallback function	write	Delegates to implementation	No Issue	Passed

SFTVault.sol

(1) Inherited contracts

(a) SFTVaultStorage: Storage contract for SFT Vault

(2) Usages

(a) using SafeMath for uint256

(b) using SafeBEP20 for IBEP20

(3) Events

(a) event Deposit(address indexed user, uint256 amount);

(b) event Withdraw(address indexed user, uint256 amount);

(c) event AdminTransfered(address indexed oldAdmin, address indexed newAdmin);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	deposit	write	Passed	No Issue	Passed
3	withdraw	write	Passed	No Issue	Passed
4	claim	write	Passed	No Issue	Passed
5	_withdraw	internal	Passed	No Issue	Passed
6	pendingFTS	read	Passed	No Issue	Passed
7	updateAndPayOutPending	internal	Passed	No Issue	Passed
8	safeFTSTransfer	internal	Passed	No Issue	Passed
9	updatePendingRewards	write	Passed	No Issue	Passed
10	updateVault	internal	Passed	No Issue	Passed
11	getAdmin	read	Passed	No Issue	Passed
12	burnAdmin	write	Passed	No Issue	Passed
13	setNewAdmin	write	Passed	No Issue	Passed
14	_become	write	Passed	No Issue	Passed
15	setFortressInfo	write	Passed	No Issue	Passed

FortressLens.sol

(1) Interface

(a) ComptrollerLensInterface: This is for comptroller Lens

(2) Structs

(a) FTokenMetadata

(b) FTokenBalances

(c) FTokenUnderlyingPrice

(d) AccountLimits

(e) GovReceipt

(f) GovProposal
(g) FTSBalanceMetadata
(h) FTSBalanceMetadataExt
(i) FortressVotes

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	fTokenMetadata	write	Passed	No Issue	Passed
2	fTokenMetadataAll	write	Infinite loop possibility	Keep array length limited	Passed
3	fTokenBalances	write	Passed	No Issue	Passed
4	fTokenBalancesAll	write	Infinite loop possibility	Keep array length limited	Passed
5	fTokenUnderlyingPrice	read	Passed	No Issue	Passed
6	fTokenUnderlyingPriceAll	read	Infinite loop possibility	Keep array length limited	Passed
7	getAccountLimits	read	Passed	No Issue	Passed
8	getGovReceipts	read	Infinite loop possibility	Keep array length limited	Passed
9	setProposal	internal	Passed	No Issue	Passed
10	getGovProposals	read	Infinite loop possibility	Keep array length limited	Passed
11	getFTSBalanceMetadata	read	Passed	No Issue	Passed
12	getFTSBalanceMetadataExt	write	Passed	No Issue	Passed
13	getFortressVotes	read	Infinite loop possibility	Keep array length limited	Passed

WhitePaperInterestRateModel.sol

(1) Inherited contracts

(a) InterestRateModel: For Interest rates

(2) Usages

(a) using SafeMath for uint

(3) Events

(a) event NewInterestParams(uint baseRatePerBlock, uint multiplierPerBlock);

(4) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	utilizationRate	read	Passed	No Issue	Passed
3	getBorrowRate	read	Passed	No Issue	Passed
4	getSupplyRate	read	Passed	No Issue	Passed

FortressPriceOracle.sol

(1) Inherited contracts

(a) PriceOracle: To get price data from market

(2) Usages

(a) using SafeMath for uint256

(3) Events

(a) event PricePosted(address asset, uint previousPriceMantissa, uint requestedPriceMantissa, uint newPriceMantissa);

(b) event NewAdmin(address oldAdmin, address newAdmin);

(4) Interface

(a) IStdReference

(5) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	getUnderlyingPrice	read	Passed	No Issue	Passed
3	setUnderlyingPrice	write	Passed	No Issue	Passed
4	setDirectPrice	write	Passed	No Issue	Passed
5	assetPrices	read	Passed	No Issue	Passed
6	compareStrings	read	Passed	No Issue	Passed
7	setAdmin	write	Passed	No Issue	Passed

FBep20Delegator.sol (This contract will be the same for all fTokens)

(1) Inherited contracts

(a) FTokenInterface: fToken functions

(b) FBep20Interface: BEP20 standard functions

(c) FDelegatorInterface: Delegator Interface

(2) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	_resignImplementation	write	Passed	No Issue	Passed
3	_setImplementation	write	Passed	No Issue	Passed
4	mint	write	Passed	No Issue	Passed
5	redeem	write	Passed	No Issue	Passed
6	redeemUnderlying	write	Passed	No Issue	Passed
7	borrow	write	Passed	No Issue	Passed
8	repayBorrow	write	Passed	No Issue	Passed
9	repayBorrowBehalf	write	Passed	No Issue	Passed
10	liquidateBorrow	write	Passed	No Issue	Passed
11	transfer	write	Passed	No Issue	Passed
12	transferFrom	write	Passed	No Issue	Passed
13	approve	write	Passed	No Issue	Passed
14	allowance	read	Passed	No Issue	Passed
15	balanceOf	read	Passed	No Issue	Passed
16	balanceOfUnderlying	write	Passed	No Issue	Passed
17	getAccountSnapshot	read	Passed	No Issue	Passed

18	borrowRatePerBlock	read	Passed	No Issue	Passed
19	supplyRatePerBlock	read	Passed	No Issue	Passed
20	totalBorrowsCurrent	write	Passed	No Issue	Passed
21	borrowBalanceCurrent	write	Passed	No Issue	Passed
22	borrowBalanceStored	read	Passed	No Issue	Passed
23	exchangeRateCurrent	write	Passed	No Issue	Passed
24	exchangeRateStored	read	Passed	No Issue	Passed
25	getCash	read	Passed	No Issue	Passed
26	accrueInterest	write	Passed	No Issue	Passed
27	seize	write	Passed	No Issue	Passed
28	_setPendingAdmin	write	Passed	No Issue	Passed
29	_setComptroller	write	Passed	No Issue	Passed
30	_setReserveFactor	write	Passed	No Issue	Passed
31	_acceptAdmin	write	Passed	No Issue	Passed
32	_addReserves	write	Passed	No Issue	Passed
33	_reduceReserves	write	Passed	No Issue	Passed
34	_transferReserves	write	Passed	No Issue	Passed
35	_setInterestRateModel	write	Passed	No Issue	Passed
36	delegateTo	internal	Passed	No Issue	Passed
37	delegateToImplementation	write	Passed	No Issue	Passed
38	delegateToViewImplementation	read	Passed	No Issue	Passed
39	delegateToViewAndReturn	read	Passed	No Issue	Passed
40	delegateAndReturn	write	Passed	No Issue	Passed
41	fallback function	write	Passed	No Issue	Passed

Timelock.sol

(1) Usages

(a) using SafeMath for uint

(2) Events

(a) event NewAdmin(address indexed newAdmin);
(b) event NewPendingAdmin(address indexed newPendingAdmin);
(c) event NewDelay(uint indexed newDelay);
(d) event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
(e) event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
(f) event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	setDelay	write	caller was required to be contract itself	It should be an admin	Passed with consent
3	acceptAdmin	write	Passed	No Issue	Passed
4	setPendingAdmin	write	caller was required to be contract itself	It should be an admin	Passed with consent
5	queueTransaction	write	Passed	No Issue	Passed
6	cancelTransaction	write	Passed	No Issue	Passed
7	executeTransaction	write	Passed	No Issue	Passed
8	getBlockTimestamp	read	Passed	No Issue	Passed

GovernorAlpha.sol

(1) Structs

(a) Proposal

(b) Receipt

(2) Events

(a) event ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description);

(b) event VoteCast(address voter, uint proposalId, bool support, uint votes);

(c) event ProposalCanceled(uint id);

(d) event ProposalQueued(uint id, uint eta);

(e) event ProposalExecuted(uint id);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	constructor	write	Passed	No Issue	Passed
2	propose	write	Passed	No Issue	Passed
3	queue	write	Passed	No Issue	Passed
4	_queueOrRevert	internal	Passed	No Issue	Passed
5	execute	write	Infinite loop possibility	Keep array length limited	Passed
6	cancel	write	Infinite loop possibility	Keep array length limited	Passed
7	getActions	read	Passed	No Issue	Passed
8	getReceipt	read	Passed	No Issue	Passed
9	state	read	Passed	No Issue	Passed
10	castVote	write	Passed	No Issue	Passed
11	castVoteBySig	write	Passed	No Issue	Passed
12	_castVote	internal	Passed	No Issue	Passed
13	__acceptAdmin	write	Passed	No Issue	Passed
14	__abdicate	write	Passed	No Issue	Passed
15	__queueSetTimelockPendingAdmin	write	Passed	No Issue	Passed

16	__executeSetTimelockPendingAdmin	write	Passed	No Issue	Passed
17	add256	read	Passed	No Issue	Passed
18	sub256	read	Passed	No Issue	Passed
19	getChainId	read	Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

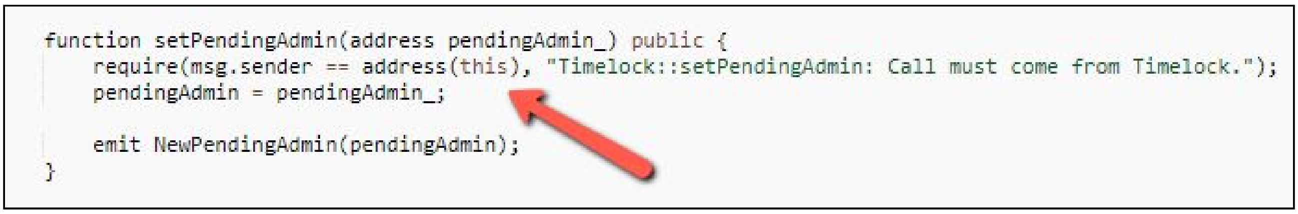
No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

(1) Caller is smart contract itself in Timelock.sol



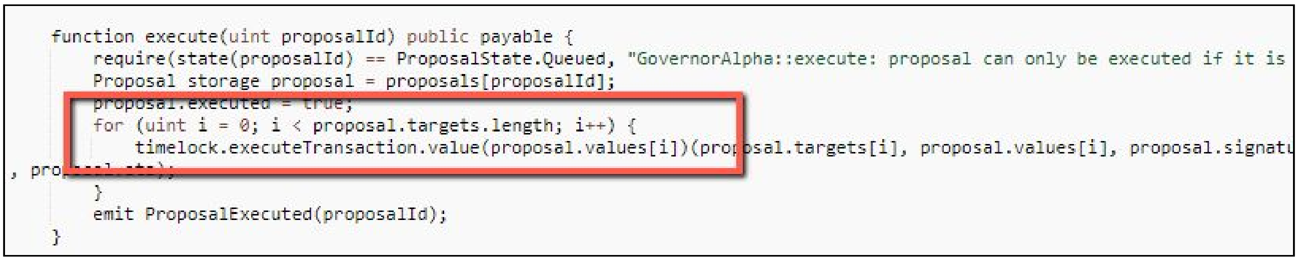
```
function setPendingAdmin(address pendingAdmin_) public {
    require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock.");
    pendingAdmin = pendingAdmin_;

    emit NewPendingAdmin(pendingAdmin);
}
```

Ideally, the caller should be an admin wallet.

Resolution: we got confirmation from the Fortress team as this is part of the plan.

Low

(1) Infinite loops possibility at multiple places:



```
    function execute(uint proposalId) public payable {
        require(state(proposalId) == ProposalState.Queued, "GovernorAlpha::execute: proposal can only be executed if it is
        Proposal storage proposal = proposals[proposalId];
        proposal.executed = true;
        for (uint i = 0; i < proposal.targets.length; i++) {
            timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i], proposal.values[i], proposal.signatu
, proposal.eta);
        }
        emit ProposalExecuted(proposalId);
    }
```

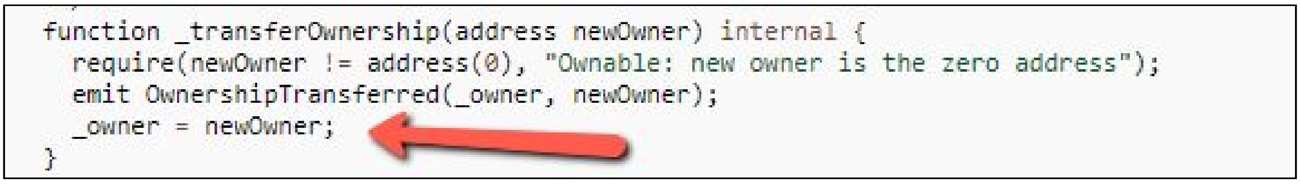
As seen in the AS-IS section, there are several places in the smart contracts, where array.length is used directly in the loops. It is recommended to put some kind of limits, so it does not go wild and create any scenario where it can hit the block gas limit.

Resolution: We got confirmation from the Fortress team that the array will be provided as limited length. And this will be taken care of from the client side.

Very Low

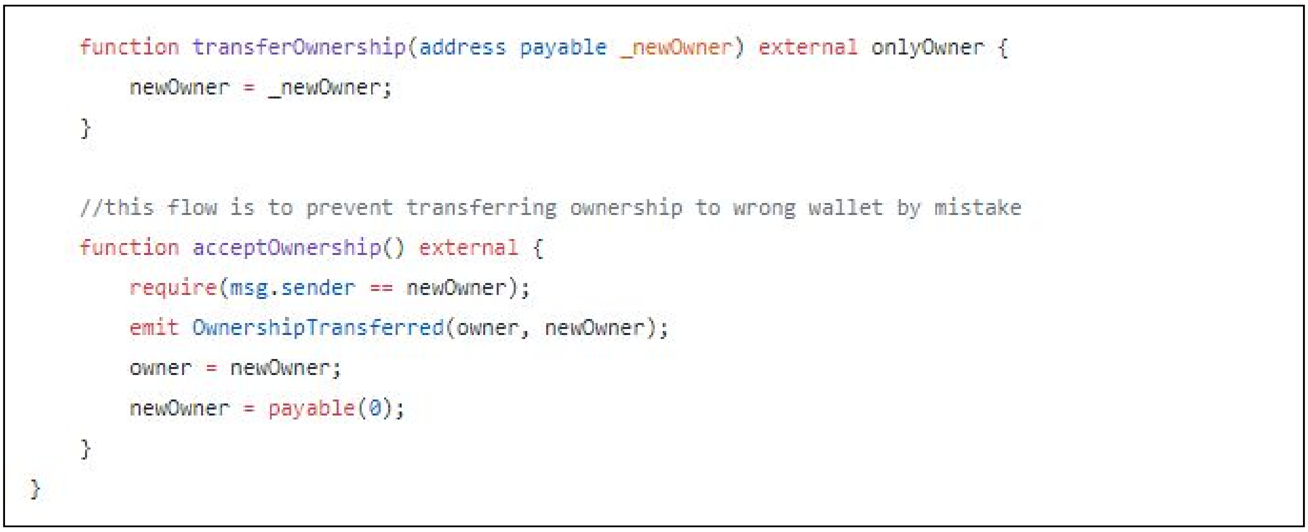
(1) Ownership transfer function:

Ownable.sol smart contract has active ownership transfer. This will be troublesome if the ownership was sent to an incorrect address by human error.



```
function _transferOwnership(address newOwner) internal {
  require(newOwner != address(0), "Ownable: new owner is the zero address");
  emit OwnershipTransferred(_owner, newOwner);
  _owner = newOwner;
}
```

so, it is a good practice to implement acceptOwnership style to prevent it. Code flow similar to below:



```
    function transferOwnership(address payable _newOwner) external onlyOwner {
        newOwner = _newOwner;
    }

    //this flow is to prevent transferring ownership to wrong wallet by mistake
    function acceptOwnership() external {
        require(msg.sender == newOwner);
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = payable(0);
    }
}
```

Resolution: Fortress team acknowledged this, as this should be taken care of from admin side.

(2) Use the latest solidity version while contract deployment to prevent any compiler version level bugs.

Resolution: This issue is acknowledged.

Discussion / Best practices:

(1) Max minting limit for FAI is not set. Also, the owner has the ability to assign other minter wallets which can mint new tokens. In an ideal scenario, the unitroller will mint it. But still there is a way to mint more tokens. This must be limited somehow to protect the tokenomics from being inflated.

(2) Overpowered functions: There are so many functions which are authorised persons (admin/owner) only. This makes this platform more centralized due to its dependence on human actions. And it would be troublesome if the private key of that owner wallet would be compromised.

(3) Approve of BEP20 standard: This can be used to front run. From the client side, only use this function to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). This should be done from the client side.

(4) getFTSAddress function in Comptroller.sol is hard coded. It is recommended to put it in a variable and also make an admin function to change this. This is useful in the event of that contract having found any bug and needing to swap that contract.

(5) FAI.sol and GovernorAlpha.sol smart contracts are having signature based actions. It is good as it saves gas cost to approve transactions in case of FAI token. But on another hand, it needs high levels of security measures in the dapps. Because if the signatures of users would have stolen, then they lose their tokens. If this is not really needed, then better to remove it as risk is far more than the benefits.

Conclusion

We were given contract code. And we have used all possible tests based on given objects as files. The contracts are written so systematic, that we did not find any major issues. **So it is good to go for the production.**

Since possible test cases can be unlimited for such extensive smart contract protocol, so we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on extensive audit procedure scope is "**Well Secured**".

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, so the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest to conduct a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Email: audit@EtherAuthority.io