Smart Contract

Security Assessment

For FITS

28 Feb 2025

Ascendant

@ascendantfi

www.ascendant.finance

Table of Contents

DISCLAIMER

This independent audit has been conducted 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 the auditor.

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 the auditor is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will the auditor 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.

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

Executive Summary

The smart contracts reviewed in this audit were found to be **Well Secured**, meaning they contain no critical severity issues that would render them too unsafe to launch. However, it is recommended that the remaining issues found within this report be resolved or mitigated to ensure best user experience.

Security Level	
Well Secured	✓
Secured	
Poorly Secured	
Insecure	

We performed an independent technical audit to identify Smart Contracts uncertainties. This shall protect the code from illegitimate authorization attempts or external & internal threats of any type. This also ensures end-to-end proofing of the contract from frauds. The audit was performed semi-manually. We analyzed the Smart Contracts code line-by-line and used an automation tool to report any suspicious code.

The following tools were used:

- Truffle
- Hardhat
- Remix IDE
- Slither
- Sol2UML

Overview

This report has been prepared for FITS for the Polygon Network. This audit provides a user-centered examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Summary

Project Name	FITS
Platform	Polygon
Language	Solidity

Contracts Assessed

Name	Location
FITsToken.sol	Not Published
FITsRewards.sol	Not Published

Findings Summary

Severity	Found
High	0
Medium	0
Low	4
Informational	19
Total	23

Classification of Issues

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.

Manual Review

Issues Checking Status

Issue Description	Checking Status
Compiler errors	PASS
Race conditions and Reentrancy. Cross-function race conditions.	PASS
Possible delays in data delivery.	PASS
Oracle calls.	PASS
Front running.	PASS
Timestamp dependence.	PASS
Integer Overflow and Underflow.	PASS
DoS with Revert.	PASS
DoS with block gas limit.	PASS
Methods execution permissions.	PASS
Economy model of the contract.	PASS
The impact of the exchange rate on the logic.	PASS
Private user data leaks.	PASS
Malicious Event log.	PASS
Scoping and Declarations.	PASS
Uninitialized storage pointers.	PASS

Arithmetic accuracy.	PASS
Design Logic.	PASS
Cross-function race conditions.	PASS
Safe Open Zeppelin contracts implementation and usage.	PASS
Fallback function security.	PASS

Severity	**Low**
Contract	FITsReward.sol
Description	Redundant Import ln:10
Code Snippet	FITs public FITsToken;
Recommendation	FITsReward depends on the import of the FITsToken contract, when it only uses mint(). Creating an interface instead of importing the entire contract will reduce the size of the compiled contract by eliminating redundancy.
Status	

Severity	**Low**
Contract	RealEstateNFT.sol
Description	Checks-Effects-Interactions: mintTo: ln 1924

Code Snippet	function mintTo(address _to, uint256 _tokenId, string memory _tokenURI, uint256 _amount) public virtual override {
	uint256 tokenIdToMint;
	uint256 nextIdToMint = nextTokenIdToMint();
	
	require(_tokenId < nextIdToMint, "invalid id");
	tokenIdToMint = _tokenId;
	
	require(USDC.transferFrom(msg.sender, address(this), tokenPrice[_tokenId]), "Failed to mint with this USDC balance");
	
	_mint(_to, tokenIdToMint, _amount, "");
	}

Recommendation	Function does not follow checks-effects-interactions flow due to an external call made during its execution. This makes the function vulnerable to reentrancy attack, but can be easily mitigated with a Reentrancy Guard.
Status	

Severity	**Low**
Contract	FITsReward.sol
Description	Code redundancy lns 88 - 96

Code Snippet	function pauseClaiming() external onlyOwner {
	require(claimingPaused == false, "FITsReward: Claiming is already paused");
	claimingPaused = true;
	}
	
	function unpauseClaiming() external onlyOwner {
	require(claimingPaused == true, "FITsReward: Claiming is already unpaused");
	claimingPaused = false;
	}

Recommendation	pauseClaiming and unpauseClaiming can be combined and controlled by a boolean input to reduce the size of the contract.
Status	

Severity	**Low**
Contract	FITsReward.sol
Description	Inflexible Dependency ln 102
Code Snippet	require(block.timestamp <= lastRootUpdatedTime + 1 days, "FITsReward: Internal Error, Please try again later");
Recommendation	The ability of the user to claim tokens is dependent on the owner manually updating the merkle root each day without fail, even when there are no new users to add to the merkle tree.
Status	

Severity	**Informational x3**
Contract	FITsReward.sol FITsToken.sol
Description	Lack of zero address check
Code Snippet	setRewardContract(address)._rewardContract setRootUpdater(address)._rootUpdater setFeeWallet(address)._feeWallet
Recommendation	Functions do not check if the addresses added are the zero address. In some scenarios the result could be loss of control or loss of tokens.
Status	

Functional Test Status

Function Name	Type/Return Type	Score
FITsReward		
claimReward	external	PASS
pauseClaiming	external	PASS
unpauseClaiming	external	PASS
setClaimCoolDown	external	PASS
setFeePercentage	external	PASS
setFeeWallet	external	PASS
setRootUpdater	external	PASS
updateMerkleRoot	external	PASS
FITs		
mint	external	PASS
setRewardContract	external	PASS
ReetrancyGuard		
_nonReentrantAfter	private	PASS
_nonReentrantBefore	private	PASS
_reentrancyGuardEntered	internal	PASS
Hashes		
_efficientKeccak256	private	PASS
communtativeKeccak256	internal	PASS

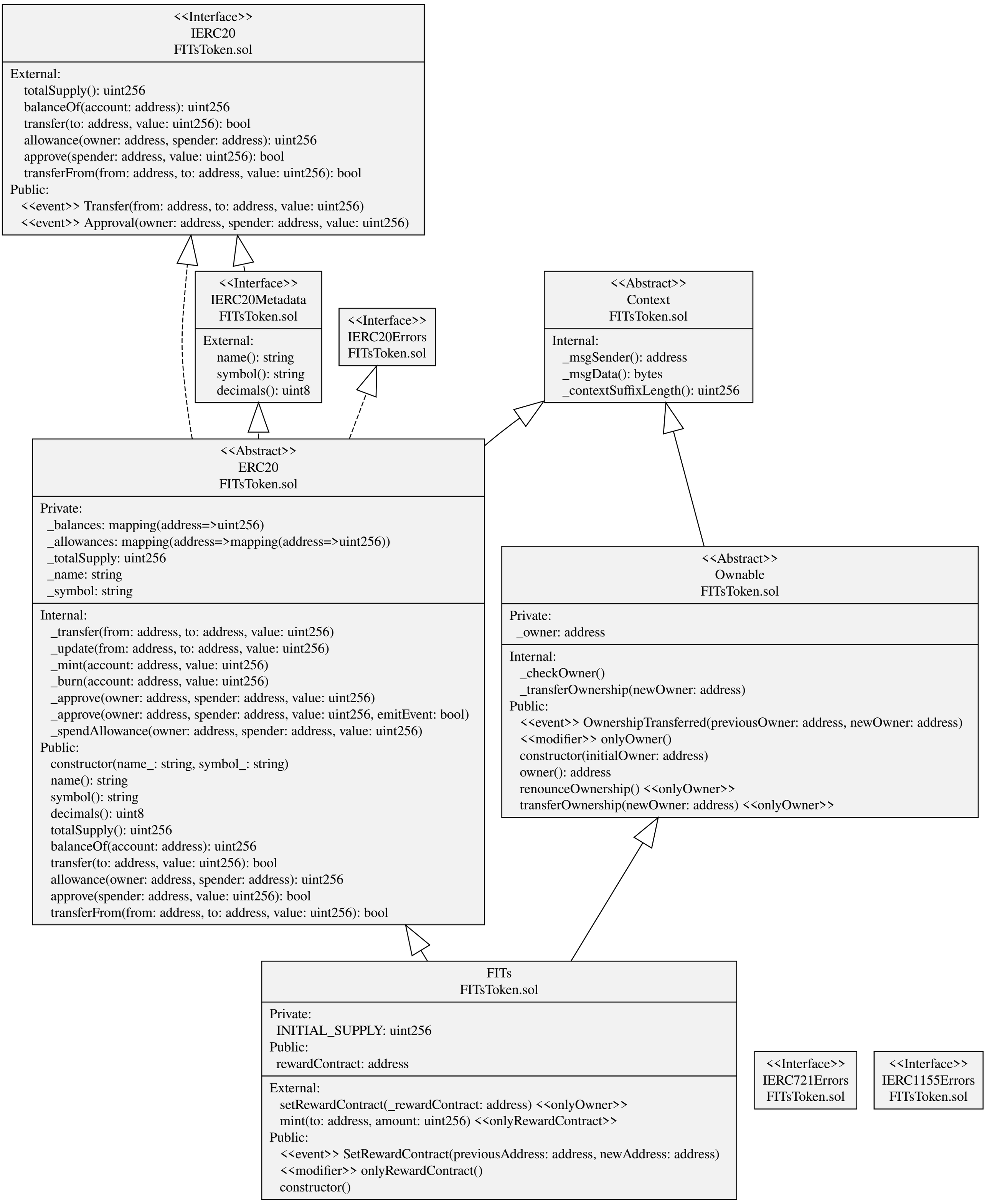
MerkleProof		
multiProofVerify	internal	PASS
multiProofVerifyCallData	internal	PASS
processMultiProof	internal	PASS
processProof	internal	PASS
processProofCallData	internal	PASS
verify	internal	PASS
verifyCalldata	internal	PASS
IERC20		
allowance	external	PASS
approve	external	PASS
balanceOf	external	PASS
totalSupply	external	PASS
transfer	external	PASS
transfeFrom	external	PASS
IERC20Metadata		
decimals	external	PASS
name	external	PASS
symbol	external	PASS
Context		
_msgData	internal	PASS
_msgSender	internal	PASS

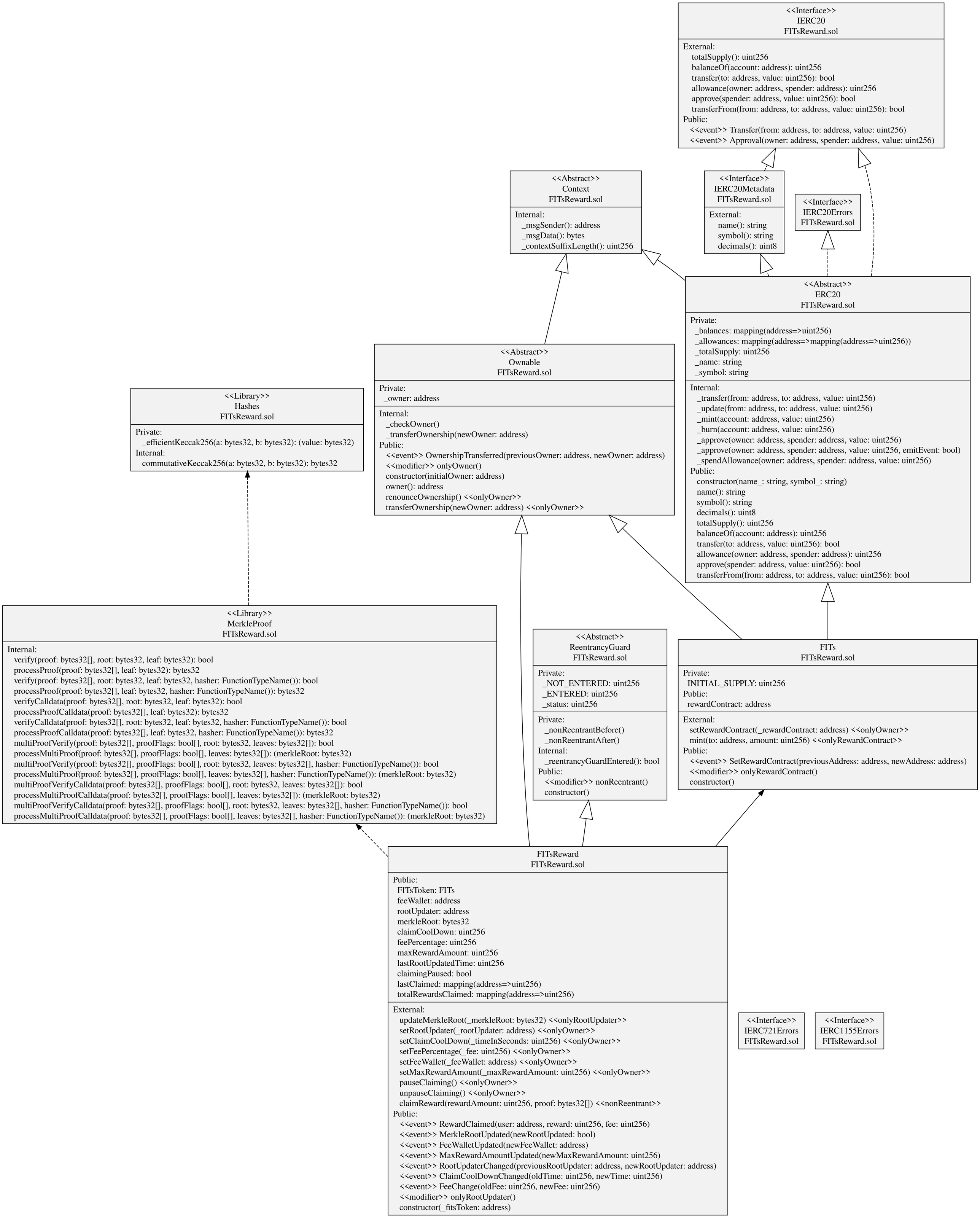
ERC20		
_afterTokenTransfer	internal	PASS
_approve	internal	PASS
_beforeTokenTransfer	internal	PASS
_burn	internal	PASS
_mint	internal	PASS
_spendAllowance	internal	PASS
_transfer	internal	PASS
allowance	public	PASS
approve	public	PASS
balanceOf	public	PASS
decreaseAllowance	public	PASS
increaseAllowance	public	PASS

Omitted Results

Note: Any issues that have been omitted from this report have been deemed by the reviewing team as irrelevant, inapplicable, and/or negligible to the proper functioning of this contract. Thus, any omitted issues can be safely ignored.

Automated Review


<<Interface>>
IERC20
FITsToken.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, value: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, value: uint256): bool
transferFrom(from: address, to: address, value: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Interface>>
IERC20Metadata
FITsToken.sol
External:
name(): string
symbol(): string
decimals(): uint8
<<Interface>>
IERC20Errors
FITsToken.sol
<<Abstract>>
Context
FITsToken.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Abstract>>
ERC20
FITsToken.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
Internal:
_transfer(from: address, to: address, value: uint256)
_update(from: address, to: address, value: uint256)
_mint(account: address, value: uint256)
_burn(account: address, value: uint256)
_approve(owner: address, spender: address, value: uint256)
_approve(owner: address, spender: address, value: uint256, emitEvent: bool)
_spendAllowance(owner: address, spender: address, value: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, value: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, value: uint256): bool
transferFrom(from: address, to: address, value: uint256): bool
<<Abstract>>
Ownable
FITsToken.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor(initialOwner: address)
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
FITs
FITsToken.sol
Private:
INITIAL_SUPPLY: uint256
Public:
rewardContract: address
External:
setRewardContract(_rewardContract: address) <<onlyOwner>>
mint(to: address, amount: uint256) <<onlyRewardContract>>
Public:
<<event>> SetRewardContract(previousAddress: address, newAddress: address)
<<modifier>> onlyRewardContract()
constructor()
<<Interface>>
IERC721Errors
FITsToken.sol
<<Interface>>
IERC1155Errors
FITsToken.sol


<<Interface>>
IERC20
FITsReward.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, value: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, value: uint256): bool
transferFrom(from: address, to: address, value: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Abstract>>
Context
FITsReward.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Interface>>
IERC20Metadata
FITsReward.sol
External:
name(): string
symbol(): string
decimals(): uint8
<<Interface>>
IERC20Errors
FITsReward.sol
<<Abstract>>
ERC20
FITsReward.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
Internal:
_transfer(from: address, to: address, value: uint256)
_update(from: address, to: address, value: uint256)
_mint(account: address, value: uint256)
_burn(account: address, value: uint256)
_approve(owner: address, spender: address, value: uint256)
_approve(owner: address, spender: address, value: uint256, emitEvent: bool)
_spendAllowance(owner: address, spender: address, value: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, value: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, value: uint256): bool
transferFrom(from: address, to: address, value: uint256): bool
<<Abstract>>
Ownable
FITsReward.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor(initialOwner: address)
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Library>>
Hashes
FITsReward.sol
Private:
_efficientKeccak256(a: bytes32, b: bytes32): (value: bytes32)
Internal:
commutativeKeccak256(a: bytes32, b: bytes32): bytes32
<<Library>>
MerkleProof
FITsReward.sol
Internal:
verify(proof: bytes32[], root: bytes32, leaf: bytes32): bool
processProof(proof: bytes32[], leaf: bytes32): bytes32
verify(proof: bytes32[], root: bytes32, leaf: bytes32, hasher: FunctionTypeName()): bool
processProof(proof: bytes32[], leaf: bytes32, hasher: FunctionTypeName()): bytes32
verifyCalldata(proof: bytes32[], root: bytes32, leaf: bytes32): bool
processProofCalldata(proof: bytes32[], leaf: bytes32): bytes32
verifyCalldata(proof: bytes32[], root: bytes32, leaf: bytes32, hasher: FunctionTypeName()): bool
processProofCalldata(proof: bytes32[], leaf: bytes32, hasher: FunctionTypeName()): bytes32
multiProofVerify(proof: bytes32[], proofFlags: bool[], root: bytes32, leaves: bytes32[]): bool
processMultiProof(proof: bytes32[], proofFlags: bool[], leaves: bytes32[]): (merkleRoot: bytes32)
multiProofVerify(proof: bytes32[], proofFlags: bool[], root: bytes32, leaves: bytes32[], hasher: FunctionTypeName()): bool
processMultiProof(proof: bytes32[], proofFlags: bool[], leaves: bytes32[], hasher: FunctionTypeName()): (merkleRoot: bytes32)
multiProofVerifyCalldata(proof: bytes32[], proofFlags: bool[], root: bytes32, leaves: bytes32[]): bool
processMultiProofCalldata(proof: bytes32[], proofFlags: bool[], leaves: bytes32[]): (merkleRoot: bytes32)
multiProofVerifyCalldata(proof: bytes32[], proofFlags: bool[], root: bytes32, leaves: bytes32[], hasher: FunctionTypeName()): bool
processMultiProofCalldata(proof: bytes32[], proofFlags: bool[], leaves: bytes32[], hasher: FunctionTypeName()): (merkleRoot: bytes32)
<<Abstract>>
ReentrancyGuard
FITsReward.sol
Private:
_NOT_ENTERED: uint256
_ENTERED: uint256
_status: uint256
Private:
_nonReentrantBefore()
_nonReentrantAfter()
Internal:
_reentrancyGuardEntered(): bool
Public:
<<modifier>> nonReentrant()
constructor()
FITs
FITsReward.sol
Private:
INITIAL_SUPPLY: uint256
Public:
rewardContract: address
External:
setRewardContract(_rewardContract: address) <<onlyOwner>>
mint(to: address, amount: uint256) <<onlyRewardContract>>
Public:
<<event>> SetRewardContract(previousAddress: address, newAddress: address)
<<modifier>> onlyRewardContract()
constructor()
FITsReward
FITsReward.sol
Public:
FITsToken: FITs
feeWallet: address
rootUpdater: address
merkleRoot: bytes32
claimCoolDown: uint256
feePercentage: uint256
maxRewardAmount: uint256
lastRootUpdatedTime: uint256
claimingPaused: bool
lastClaimed: mapping(address=>uint256)
totalRewardsClaimed: mapping(address=>uint256)
External:
updateMerkleRoot(_merkleRoot: bytes32) <<onlyRootUpdater>>
setRootUpdater(_rootUpdater: address) <<onlyOwner>>
setClaimCoolDown(_timeInSeconds: uint256) <<onlyOwner>>
setFeePercentage(_fee: uint256) <<onlyOwner>>
setFeeWallet(_feeWallet: address) <<onlyOwner>>
setMaxRewardAmount(_maxRewardAmount: uint256) <<onlyOwner>>
pauseClaiming() <<onlyOwner>>
unpauseClaiming() <<onlyOwner>>
claimReward(rewardAmount: uint256, proof: bytes32[]) <<nonReentrant>>
Public:
<<event>> RewardClaimed(user: address, reward: uint256, fee: uint256)
<<event>> MerkleRootUpdated(newRootUpdated: bool)
<<event>> FeeWalletUpdated(newFeeWallet: address)
<<event>> MaxRewardAmountUpdated(newMaxRewardAmount: uint256)
<<event>> RootUpdaterChanged(previousRootUpdater: address, newRootUpdater: address)
<<event>> ClaimCoolDownChanged(oldTime: uint256, newTime: uint256)
<<event>> FeeChange(oldFee: uint256, newFee: uint256)
<<modifier>> onlyRootUpdater()
constructor(_fitsToken: address)
<<Interface>>
IERC721Errors
FITsReward.sol
<<Interface>>
IERC1155Errors
FITsReward.sol

Conclusion

The smart contracts reviewed in this audit contain no critical severity issues and all Medium to Low issues have either been corrected or acknowledged.

Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.

Ascendant

Ascendant

@ascendantfi

www.ascendant.finance

Ascendant