InterFi NETWORK

@INTERFINETWORK

SMART CONTRACT SECURITY AUDIT OF

REMATIC FINANCE

SMART CONTRACT AUDIT | SOLIDITY DEVELOPMENT & TESTING | PROJECT EVALUATION

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN

Audit Introduction

Auditing Firm	InterFi Network
Audit Architecture	InterFi Echelon Auditing Standard
Language	Solidity
Client Firm	Rematic Finance
Website	https://rematic.finance/
Telegram	https://t.me/rematicfinance/
Twitter	https://twitter.com/RematicFinance/
YouTub	https://www.youtube.com/channel/UC10S4DkfEgMuEWZ09GhVo-Q/ https://www.youtube.com/c/LSPTRADING/
Discord	https://discord.gg/nBj4TEHbRa/
Report Date	July 16, 2022

About Rematic Finance

RFTX Will Serve as The Token for The Rematic Finance Services Platform. Services Will Include Staking (Single Asset and Lending for Yield), Collateralized Borrowing, 401k, And Pension Funds. Earnings Will Be Generated Through A Diversified and Proprietary Risk Adjusted Farming Portfolio. To Bridge Real-World Finance and Cryptocurrency, RFTX Will Allow Collateralization of Tangible Goods and Property Along With Crypto Assets.

$RFTX Is Itself a Tokenomics Based Reflection Token. Reflections Will Be Provided Which Will Allow Investors to opt Into Long Term Savings Protocols That Generate A Truly Tailored Passive Income Stream.

Audit Summary

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- Rematic Finance's solidity source code has **LOW RISK SEVERITY**
- Rematic Finance's smart contract has an **ACTIVE OWNERSHIP**
- Rematic Finance's centralization risk correlated to the active owner is **HIGH**
- Important owner privileges – **AUTHORIZE UPGRADE, SET ADMIN, SET FEES, SET MAX TRANSFER LIMIT, SET TIME BUYS AND SELLS**
- Rematic Finance's smart contract has an upgradability mechanism. The smart contract utilizes EIP-1822: Universal Upgradeable Proxy Standard **UUPS**. Functions included in Rematic.sol smart contract can perform an upgrade of an {ERC1967Proxy}, when this contract is set as the implementation behind such a proxy.

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. For a detailed understanding of risk severity, source code vulnerability, exploitability, and audit disclaimer, kindly refer to the audit.

🚨 Token Contract address: **0x9039404e289ADA422A293665f94c1CFf3Ff174e0**

🚨 Proxy Contract address: **0xebA996ad44A3Aef432a853D2D42f6bD30Bc7c990**

⛓️ Blockchain: **Binance Smart Chain**

✅ Verify the authenticity of this report on InterFi's GitHub: **https://github.com/interfinetwork**

Table Of Contents

Audit Scope

InterFi was consulted by Rematic Finance to conduct the smart contract security audit of their solidity source codes. The audit scope of work is strictly limited to the mentioned solidity file(s) only:

- Rematic.sol
- ERC1967Proxy.sol

Solidity Source Codes On Blockchain (Verified Contract Source Code)

https://bscscan.com/address/0x9039404e289ada422a293665f94c1cff3ff174e0#code

Contract Name: Rematic

Compiler Version: v0.8.4

Optimization Enabled: Yes with 1000 runs

https://bscscan.com/address/0xebA996ad44A3Aef432a853D2D42f6bD30Bc7c990#code

Contract Name: ERC1967Proxy

Compiler Version: v0.8.2

Optimization Enabled: Yes with 200 runs

SHA-1 Hash

Solidity source code is audited at hash #6218656691149c4b1b356d45793e4e83e3ad31a4

Audit Methodology

The scope of this report is to audit the smart contract source code of Rematic Finance. InterFi has scanned contracts and reviewed codes for common vulnerabilities, exploits, hacks, and back-doors. Due to being out of scope, InterFi has not tested contracts on testnet to assess any functional flaws. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Category	
Smart Contract Vulnerabilities	❖ Re-entrancy ❖ Unhandled Exceptions ❖ Transaction Order Dependency ❖ Integer Overflow ❖ Unrestricted Action ❖ Incorrect Inheritance Order ❖ Typographical Errors ❖ Requirement Violation ❖ Gas Limit and Loops
Source Code Review	❖ Deployment Consistency ❖ Repository Consistency ❖ Data Consistency ❖ Token Supply Manipulation ❖ Access Control and Authorization ❖ Operations Trail and Event Generation ❖ Assets Manipulation ❖ Ownership Control ❖ Liquidity Access

InterFi's Echelon Audit Standard

The aim of InterFi's "Echelon" standard is to analyze smart contracts and identify the vulnerabilities and the hacks. Kindly note, InterFi does not test smart contracts on testnet. It is recommended that smart contracts are thoroughly tested prior to the audit submission. Mentioned are the steps used by InterFi to audit smart contracts:

1. Solidity smart contract source code reviewal:
   - ❖ Review of the specifications, sources, and instructions provided to InterFi to make sure we understand the size, and scope of the smart contract audit.
   - ❖ Manual review of code, which is the process of reading source code line-by-line to identify potential vulnerabilities.
2. Static, Manual, and Software analysis:
   - ❖ Test coverage analysis is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
   - ❖ Symbolic execution is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts

Automated 3P frameworks used to assess the smart contract vulnerabilities

- ❖ Consensys Tools
- ❖ SWC Registry
- ❖ Solidity Coverage
- ❖ Open Zeppelin Code Analyzer
- ❖ Solidity Code Complier

Risk Classification

Smart contracts are generally designed to manipulate and hold funds denominated in ETH/BNB. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract. Below are the typical risk levels of a smart contract:

Vulnerable: A contract is vulnerable if it has been flagged by a static analysis tool as such. As we will see later, this means that some contracts may be vulnerable because of a false positive.

Exploitable: A contract is exploitable if it is vulnerable and the vulnerability could be exploited by an external attacker. For example, if the "vulnerability" flagged by a tool is in a function that requires owning the contract, it would be vulnerable but not exploitable.

Exploited: A contract is exploited if it received a transaction on the main network which triggered one of its vulnerabilities. Therefore, a contract can be vulnerable or even exploitable without having been exploited.

Risk severity	Meaning
! High	This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
! Medium	This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity
! Low	This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.
! Informational	This level vulnerabilities can be ignored. They are code style violations and informational statements in the code. They may not affect the smart contract execution

Centralization Risk

Centralization risk is the most common cause of decentralized finance hacks. When a smart contract has an active contract ownership, the risk related to centralization is elevated. There are some well-intended reasons to be an active contract owner, such as:

- Contract owner can be granted the power to `pause()` or `lock()` the contract in case of an external attack.
- Contract owner can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale, and to list on an exchange.

Authorizing a full centralized power to a single body can be dangerous. Unfortunately, centralization related risks are higher than common smart contract vulnerabilities. Centralization of ownership creates a risk of rug pull scams, where owners cash out tokens in such quantities that they become valueless. **Most important question to ask here is, how to mitigate centralization risk?** Here's InterFi's recommendation to lower the risks related to centralization hacks:

- Smart contract owner's private key must be carefully secured to avoid any potential hack.
- Smart contract ownership should be shared by multi-signature (multi-sig) wallets.
- Smart contract ownership can be locked in a contract, user voting, or community DAO can be introduced to unlock the ownership.

Rematic Finance's Centralization Status

- Rematic Finance's smart contract has an **active ownership.**
- Smart contract ownership is set to **0xee83427c574b4b2646a23d00880aa1f3142b8975** at the time of the audit.

Static Analysis

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable
🔒	Function is locked
🔐	Function can be accessed
!	Important functionality

Rematic	Implementation	ERC20BurnableUpgradeable, UUPSUpgradeable, OwnableUpgradeable		
└	initialize	Public !	🛑	initializer
└	_authorizeUpgrade	Internal 🔒	🛑	onlyOwner
└	_basicTransfer	Internal 🔒	🛑	
└	_takeFee	Internal 🔒	🛑	
└	_isOnSwap	Internal 🔒	🛑	
└	_checkAntiBot	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	antiWhale
└	_updateDivBalances	Internal 🔒	🛑	
└	setAdminContractAdddress	Public !	🛑	onlyOwner
└	setBurnWallet	Public !	🛑	onlyRematicFinanceAdmin
└	setStakingWallet	Public !	🛑	onlyRematicFinanceAdmin
└	setTxFeeRate	Public !	🛑	onlyRematicFinanceAdmin
└	setBurnFeeRate	Public !	🛑	onlyRematicFinanceAdmin
└	setStakingFeeRate	Public !	🛑	onlyRematicFinanceAdmin
└	setIsOnBurnFee	Public !	🛑	onlyRematicFinanceAdmin
└	setIsOnStakingFee	Public !	🛑	onlyRematicFinanceAdmin
└	totalCirculatingSupply	Public !		NO !
└	isExcludedFromAntiwhale	Public !		NO !
└	maxTransferAmount	Public !		NO !
└	setAutomatedMarketMakerPair	Public !	🛑	onlyOwner
└	_setAutomatedMarketMakerPair	Private 🔐	🛑	
└	excludeFromFees	Public !	🛑	onlyOwner
└	isExcludedFromFees	Public !		NO !
└	_isExcludedFromFees	Internal 🔒		
└	withdrawToken	Public !	🛑	onlyOwner
└	widthrawBNB	Public !	🛑	onlyOwner
└	excludeFromAntiwhale	Public !	🛑	onlyOwner
└	excludedFromAntiBot	Public !	🛑	onlyOwner
└	isExcludedFromAntiBot	Public !		NO !
└	changeTimeSells	Public !	🛑	onlyOwner

└	changeTimeBuys	Public ❗	🛑	onlyOwner
└	setMaxTransfertAmountRate	Public ❗	🛑	onlyOwner
└	setTradeOn	Public ❗	🛑	onlyOwner
└	_sendToAdminContractForLiquidation	Internal 🔒	🛑	
└	SetSwapThreshold	Public ❗	🛑	onlyOwner
ERC20Upgradeable	Implementation	Initializable, ContextUpgradeable, IERC20Upgradeable, IERC20MetadataUpgradeable		
└	__ERC20_init	Internal 🔒	🛑	onlyInitializing
└	__ERC20_init_unchained	Internal 🔒	🛑	onlyInitializing
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_spendAllowance	Internal 🔒	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_afterTokenTransfer	Internal 🔒	🛑	
ERC20BurnableUpgradeable	Implementation	Initializable, ContextUpgradeable, ERC20Upgradeable		
└	__ERC20Burnable_init	Internal 🔒	🛑	onlyInitializing
└	__ERC20Burnable_init_unchained	Internal 🔒	🛑	onlyInitializing
└	burn	Public ❗	🛑	NO❗
└	burnFrom	Public ❗	🛑	NO❗
OwnableUpgradeable	Implementation	Initializable, ContextUpgradeable		
└	__Ownable_init	Internal 🔒	🛑	onlyInitializing
└	__Ownable_init_unchained	Internal 🔒	🛑	onlyInitializing
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
UUPSUpgradeable	Implementation	Initializable, IERC1822ProxiableUpgradeable, ERC1967UpgradeUpgradeable		
└	__UUPSUpgradeable_init	Internal 🔒	🛑	onlyInitializing
└	__UUPSUpgradeable_init_unchained	Internal 🔒	🛑	onlyInitializing
└	proxiableUUID	External ❗		notDelegated
└	upgradeTo	External ❗	🛑	onlyProxy

└	upgradeToAndCall	External ❗	💵	onlyProxy
└	_authorizeUpgrade	Internal 🔒	🛑	
IRematicAdmin	Interface			
└	setBalance	External ❗	🛑	NO❗
└	recordTransactionHistoryForHoldersPartition	External ❗	🛑	NO❗
└	startLiquidate	External ❗	🛑	NO❗
└	pancakeSwapPair	External ❗	🛑	NO❗
└	pancakeSwapRouter02Address	External ❗	🛑	NO❗
└	_excludeFromDividendsByRematic	External ❗	🛑	NO❗
IERC20	Interface			
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transfer	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
IERC20Upgradeable	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
IERC20MetadataUpgradeable	Interface	IERC20Upgradeable		
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
ContextUpgradeable	Implementation	Initializable		
└	__Context_init	Internal 🔒	🛑	onlyInitializing
└	__Context_init_unchained	Internal 🔒	🛑	onlyInitializing
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Initializable	Implementation			
└	_isConstructor	Private 🔐		
AddressUpgradeable	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	

└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	verifyCallResult	Internal 🔒		
IERC1822ProxiableUpgradeable	Interface				
└	proxiableUUID	External ❗		NO❗
ERC1967UpgradeUpgradeable	Implementation	Initializable			
└	__ERC1967Upgrade_init	Internal 🔒	🛑	onlyInitializing
└	__ERC1967Upgrade_init_unchained	Internal 🔒	🛑	onlyInitializing
└	_getImplementation	Internal 🔒		
└	_setImplementation	Private 🔐	🛑	
└	_upgradeTo	Internal 🔒	🛑	
└	_upgradeToAndCall	Internal 🔒	🛑	
└	_upgradeToAndCallUUPS	Internal 🔒	🛑	
└	_getAdmin	Internal 🔒		
└	_setAdmin	Private 🔐	🛑	
└	_changeAdmin	Internal 🔒	🛑	
└	_getBeacon	Internal 🔒		
└	_setBeacon	Private 🔐	🛑	
└	_upgradeBeaconToAndCall	Internal 🔒	🛑	
└	_functionDelegateCall	Private 🔐	🛑	
IBeaconUpgradeable	Interface				
└	implementation	External ❗		NO❗
StorageSlotUpgradeable	Library				
└	getAddressSlot	Internal 🔒		
└	getBooleanSlot	Internal 🔒		
└	getBytes32Slot	Internal 🔒		
└	getUint256Slot	Internal 🔒		

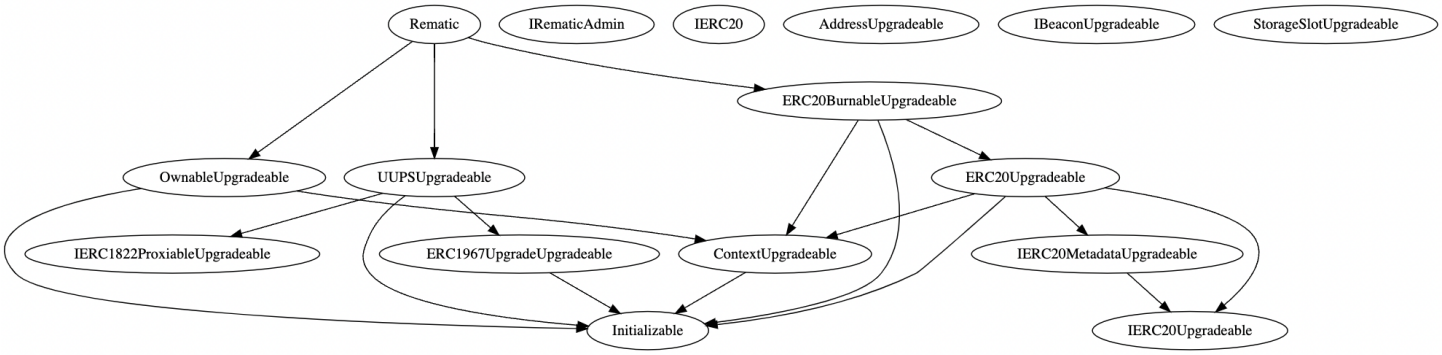
Software Analysis

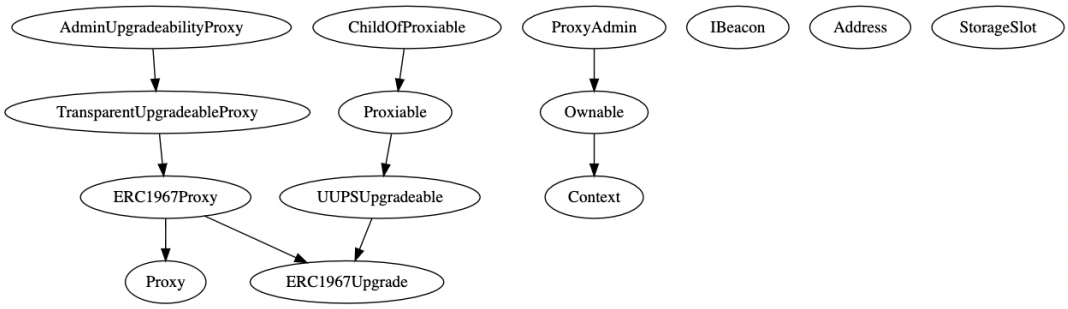
Function Signatures

```
20d355e5  =>  setAdminContractAdddress(address)
1c4ba3ed  =>  setBurnWallet(address)
1a860c3e  =>  setStakingWallet(address)
67647e43  =>  setTxFeeRate(uint256)
5dfd8b53  =>  setBurnFeeRate(uint256)
468b4f10  =>  setStakingFeeRate(uint256)
5ff80925  =>  setIsOnBurnFee(bool)
4bc1fcb7  =>  setIsOnStakingFee(bool)
5ee0ce31  =>  totalCirculatingSupply()
09bb0732  =>  isExcludedFromAntiwhale(address)
a9e75723  =>  maxTransferAmount()
9a7a23d6  =>  setAutomatedMarketMakerPair(address,bool)
a7f7b36f  =>  _setAutomatedMarketMakerPair(address,bool)
c0246668  =>  excludeFromFees(address,bool)
4fbee193  =>  isExcludedFromFees(address)
e0bf7fd1  =>  _isExcludedFromFees(address)
3aeac4e1  =>  withdrawToken(address,address)
b25fbd2c  =>  widthrawBNB(address)
e2becf02  =>  excludeFromAntiwhale(address,bool)
aee06660  =>  excludedFromAntiBot(address,bool)
5d5b29c1  =>  isExcludedFromAntiBot(address)
600d93f6  =>  excludFromFee(address,bool)
5342acb4  =>  isExcludedFromFee(address)
7dc0bd01  =>  changeTimeSells(uint256)
ccafb604  =>  changeTimeBuys(uint256)
336bb66b  =>  setMaxTransfertAmountRate(uint256)
a523ab6a  =>  setTradeOn(bool)
5e2883d2  =>  _sendToAdminContractForLiquidation()
dd535c8a  =>  SetSwapThreshold(uint256)
678bd718  =>  __ERC20_init(string,string)
46753fdb  =>  __ERC20_init_unchained(string,string)
06fdde03  =>  name()
95d89b41  =>  symbol()
313ce567  =>  decimals()
18160ddd  =>  totalSupply()
70a08231  =>  balanceOf(address)
a9059cbb  =>  transfer(address,uint256)
dd62ed3e  =>  allowance(address,address)
095ea7b3  =>  approve(address,uint256)
23b872dd  =>  transferFrom(address,address,uint256)
a457c2d7  =>  decreaseAllowance(address,uint256)
4e6ec247  =>  _mint(address,uint256)
6161eb18  =>  _burn(address,uint256)
104e81ff  =>  _approve(address,address,uint256)
1532335e  =>  _spendAllowance(address,address,uint256)
```

```
cad3be83  =>  _beforeTokenTransfer(address,address,uint256)
8f811a1c  =>  _afterTokenTransfer(address,address,uint256)
0fa5741a  =>  __ERC20Burnable_init()
f57c5ec1  =>  __ERC20Burnable_init_unchained()
42966c68  =>  burn(uint256)
79cc6790  =>  burnFrom(address,uint256)
0142eb11  =>  __Ownable_init()
5ce29e24  =>  __Ownable_init_unchained()
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
d29d44ee  =>  _transferOwnership(address)
6e7fd379  =>  __UUPSUpgradeable_init()
ce8a1477  =>  __UUPSUpgradeable_init_unchained()
52d1902d  =>  proxiableUUID()
3659cfe6  =>  upgradeTo(address)
4f1ef286  =>  upgradeToAndCall(address,bytes)
e8a6a289  =>  setBalance(address,uint256,uint256)
7ac4f2ca  =>  recordTransactionHistoryForHoldersPartition(address,uint256,bool)
6cac5400  =>  startLiquidate()
429494f2  =>  pancakeSwapPair()
3786987a  =>  pancakeSwapRouter02Address()
a836fe0d  =>  _excludeFromDividendsByRematic(address)
f08d647e  =>  __Context_init()
ab96f671  =>  __Context_init_unchained()
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
45549c1f  =>  _isConstructor()
24a084df  =>  sendValue(address,uint256)
a0b5ffb0  =>  functionCall(address,bytes)
241b5886  =>  functionCall(address,bytes,string)
2a011594  =>  functionCallWithValue(address,bytes,uint256)
d525ab8a  =>  functionCallWithValue(address,bytes,uint256,string)
c21d36f3  =>  functionStaticCall(address,bytes)
dbc40fb9  =>  functionStaticCall(address,bytes,string)
946b5793  =>  verifyCallResult(bool,bytes,string)
d69dce32  =>  __ERC1967Upgrade_init()
0f039eff  =>  __ERC1967Upgrade_init_unchained()
42404e07  =>  _getImplementation()
bb913f41  =>  _setImplementation(address)
267b04ae  =>  _upgradeToAndCall(address,bytes,bool)
d7a9f039  =>  _upgradeToAndCallUUPS(address,bytes,bool)
839f5fb8  =>  _getAdmin()
3a74a767  =>  _setAdmin(address)
353dfc01  =>  _changeAdmin(address)
2bad8ba0  =>  _getBeacon()
073d36b4  =>  _setBeacon(address)
9ba186fe  =>  _upgradeBeaconToAndCall(address,bytes,bool)
378f61a0  =>  _functionDelegateCall(address,bytes)
5c60da1b  =>  implementation()
```


Inheritance Graph





Manual Analysis

Function	Description	Available	Status
Total Supply	provides information about the total token supply	Yes	**Passed**
Balance Of	provides account balance of the owner's account	Yes	**Passed**
Transfer	executes transfers of a specified number of tokens to a specified address	Yes	**Passed**
Approve	allow a spender to withdraw a set number of tokens from a specified account	Yes	**Passed**
Allowance	returns a set number of tokens from a spender to the owner	Yes	**Passed**
Burn	executes transfers of a specified number of tokens to a burn address	Yes	**Passed**
Staking	executes transfers of a specified staking reward token to a specified address	Yes	**Passed**
Max Transfer	a non-whitelisted wallet can only transfer a specified number of tokens	Yes	**! Low**
Contract Fees	executes fee collection from swap events and/or transfer events	Yes	**! Low**
Cooldown Timer	functionality to limit the number of transactions that a wallet can make within 24-hours	Yes	**Passed**
Anti Bot	stops some or all bot wallets from interacting with the smart contract	Yes	**Passed**

Function	Description	Available	Status
Transfer Ownership	executes transfer of contract ownership to a specified wallet	Yes	**Passed**
Renounce Ownership	executes transfer of contract ownership to a dead address	Yes	**Passed**

Notable Information

- Smart contract has an **upgradability mechanism**. The smart contract utilizes EIP-1822: Universal Upgradeable Proxy Standard **UUPS**. Functions included in Rematic.sol smart contract can perform an upgrade of an {ERC1967Proxy}, when this contract is set as the implementation behind such a proxy.

```
contract Rematic is ERC20BurnableUpgradeable, UUPSUpgradeable, OwnableUpgradeable {
```

- Smart contract owner can **change trading status.** This function module can be used to stop users from buying, and selling assets.

```
    function setTradeOn(bool flag) public onlyOwner {
        require(tradeOn != flag, "Same value set already");
        tradeOn = flag;
```

- Smart contract owner can **authorize to be contract admin.** Authorized wallet to modify "write contract" parameters. When ownership is transferred, previous privileges should not remain authorized.

```
    function setAdminContractAdddress(address _address) public onlyOwner {
        require(_address != address(adminContract), "RFTX: The adminContract already has that
address");
    function _setAdmin(address newAdmin) private {
        require(newAdmin != address(0), "ERC1967: new admin is the zero address");
        StorageSlotUpgradeable.getAddressSlot(_ADMIN_SLOT).value = newAdmin;
    function _changeAdmin(address newAdmin) internal {
        emit AdminChanged(_getAdmin(), newAdmin);

 ADMIN CONTRACT IN CODE: 0xF555A2D0744dd53906A369AfcF8f985C4a32B0dE
 UPDATED ADMIN CONTRACT: 0x85298d5a9d8f54c45f3454813cbffa944d4692d1
```

- When contract uses **active ownable and upgradable mechanisms,** smart contract callers / callee addresses and functions can be accessed, modified, altered, locked, and unlocked at the contract owner's peril.

- Smart contract utilizes **redundant code** for `transferOwnership()`. Ideal transfer ownership code should look be written like:

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
```

- Smart contract owner can approve token **burn** to reduce the circulating supply.
- Smart contract utilizes **antibot** function module to allow limited transactions for a number of blocks. `changeTimeSells()` and `changeTimeBuys()` are used to achieve antibot logic.

```
function changeTimeSells(uint _value) public onlyOwner {
    require(_value <= 60 * 60 * 60, "Max 1 hour");
    timeBetweenSells = _value;
function changeTimeBuys(uint _value) public onlyOwner {
    require(_value <= 60 * 60 * 60, "Max 1 hour");
    timeBetweenBuys = _value;
```

- Smart contract owner can **change transaction fees**. This function module can be used to impose extraordinary fees. No arbitrary limit set.


```
function setTxFeeRate(uint256 _newValue) public onlyRematicFinanceAdmin {
    require(_newValue != txFeeRate, "RFTX Admin: already same value");
function setBurnFeeRate(uint256 _newValue) public onlyRematicFinanceAdmin {
    require(_newValue != burnFeeRate, "RFTX Admin: already same value");
function setStakingFeeRate(uint256 _newValue) public onlyRematicFinanceAdmin {
    require(_newValue != stakingFeeRate, "RFTX Admin: already same value");
```

- Smart contract owner can **change max transfer limit.** The smart contract owner can change the value to “zero”. No arbitrary limit set.

```
function setMaxTransfertAmountRate(uint256 value) public onlyOwner {
    require(value > 0, "fail");
```

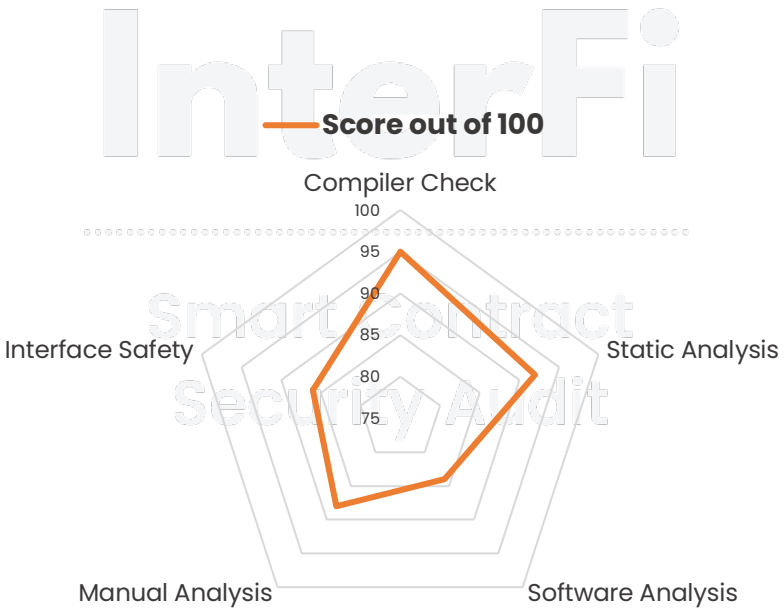
SWC Attacks

SWC ID	Description	Status
SWC-101	Integer Overflow and Underflow	**Passed**
SWC-102	Outdated Compiler Version	**! Informational**
SWC-103	Floating Pragma	**Passed**
SWC-104	Unchecked Call Return Value	**Passed**
SWC-105	Unprotected Ether Withdrawal	**Passed**
SWC-106	Unprotected SELF-DESTRUCT Instruction	**Passed**
SWC-107	Re-entrancy	**Passed**
SWC-108	State Variable Default Visibility	**Passed**
SWC-109	Uninitialized Storage Pointer	**Passed**
SWC-110	Assert Violation	**Passed**
SWC-111	Use of Deprecated Solidity Functions	**Passed**
SWC-112	Delegate Call to Untrusted Callee	**Passed**
SWC-113	DoS with Failed Call	**Passed**
SWC-114	Transaction Order Dependence	**Passed**
SWC-115	Authorization through tx.origin	**Passed**
SWC-116	Block values as a proxy for time	**Passed**
SWC-117	Signature Malleability	**Passed**
SWC-118	Incorrect Constructor Name	**Passed**

SWC-119	Shadowing State Variables	**Passed**
SWC-120	Weak Sources of Randomness from Chain Attributes	**Passed**
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**
SWC-123	Requirement Violation	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**
SWC-129	Typographical Error	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**
SWC-131	Presence of unused variables	**Passed**
SWC-132	Unexpected Ether balance	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**
SWC-134	Message call with the hardcoded gas amount	**Passed**
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**! Informational**
SWC-136	Unencrypted Private Data On-Chain	**Passed**

Risk Status & Radar Chart

Risk Severity	Status
High	No high severity issues identified
Medium	No medium severity issues identified
Low	2 low severity issues identified
Informational	2 informational severity issues identified
Centralization Risk	Active contract ownership identified



Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- Rematic Finance's smart contract source code has **LOW RISK SEVERITY**
- Rematic Finance's smart contract has an **ACTIVE OWNERSHIP**
- Rematic Finance's centralization risk correlated to the active owner is **HIGH**

Note for stakeholders

- Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security.
- If the smart contract is not deployed on any blockchain at the time of the audit, the contract can be modified or altered before blockchain development. Verify contract's deployment status in the audit report.
- Make sure that the project team's KYC/identity is verified by an independent firm.
- Always check if the contract's liquidity is locked. A longer liquidity lock plays an important role in the project's longevity. It is recommended to have multiple liquidity providers.
- Examine the unlocked token supply in the owner, developer, or team's private wallets. Understand the project's tokenomics, and make sure the tokens outside of the LP Pair are vested or locked for a longer period.

Important Disclaimer

InterFi Network provides contract development, testing, auditing and project evaluation services for blockchain projects. The purpose of the audit is to analyze the on-chain smart contract source code and to provide a basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purpose without InterFi's prior written consent.**

InterFi provides the easy-to-understand assessment of the project, and the smart contract (otherwise known as the source code). The audit makes no statements or warranties on the security of the code. It also cannot be considered as enough assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have used all the data at our disposal to provide the transparent analysis, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. **Be aware that smart contracts deployed on a blockchain aren't resistant to external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security. Therefore, InterFi does not guarantee the explicit security of the audited smart contract.**

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

This report should not be considered as an endorsement or disapproval of any project or team. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. Do conduct your due diligence and consult your financial advisor before making any investment decisions.

About InterFi Network

InterFi Network provides intelligent blockchain solutions. InterFi is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **InterFi's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy to use.**

InterFi is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. **InterFi provides manual, static, and automatic smart contract analysis, to ensure that project is checked against known attacks and potential vulnerabilities.**

To learn more, visit **https://interfi.network**

To view our audit portfolio, visit **https://github.com/interfinetwork**

To book an audit, message **https://t.me/interfiaudits**

@INTERFINETWORK

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA