InterFi NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

APPLESWAP AI - ROUTER CONTRACT

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	AppleSwap AI
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	
Blockchain	
Centralization	Active ownership
Commit	2c4140f049f0e128c3832f2e99ce2745213a89e3
Website	https://appleswap.ai/#/home
Telegram	https://t.me/AppleSwapAI_Ann/ | https://t.me/appleswapai_global/
Twitter	https://twitter.com/AppleSwapAI/
Medium	https://medium.com/@AppleSwap.AI/
Report Date	August 09, 2023

Verify the authenticity of this report on our website: https://www.github.com/interfinetwork

EXECUTIVE SUMMARY

InterFi has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical 🔴	Major 🟠	Medium 🟡	Minor 🟢	Unknown 🟤
Open	0	1	0	4	1
Acknowledged	0	1	0	2	0
Resolved	0	0	0	0	0
Noteworthy Privileges	Set Dex Token, Set Swap Fee, Set Team Wallet, Set Amount Get Free Tax				

ℹ️ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

ℹ️ Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

TABLE OF CONTENTS

SCOPE OF WORK

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

- AppleSwapAI.sol

If source codes are not deployed on the main net, they can be modified or altered before main-net deployment.

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of InterFi's auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major 🟠	These risks 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.
Medium 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles 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 privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💵	Function is payable
🔒	Function is internal
🔐	Function is private
❗	Function is important

```
| **Context** | Implementation |  |||
| └ | _msgSender | Internal 🔒 |   | |
| └ | _msgData | Internal 🔒 |   | |
||||||
| **Ownable** | Implementation | Context |||
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | owner | Public ❗ |   |NO❗ |
| └ | _checkOwner | Internal 🔒 |   | |
| └ | renounceOwnership | Public ❗ | 🛑  | onlyOwner |
| └ | transferOwnership | Public ❗ | 🛑  | onlyOwner |
| └ | _transferOwnership | Internal 🔒 | 🛑  | |
||||||
| **IERC20** | Interface |  |||
| └ | totalSupply | External ❗ |   |NO❗ |
| └ | balanceOf | External ❗ |   |NO❗ |
| └ | transfer | External ❗ | 🛑  |NO❗ |
| └ | allowance | External ❗ |   |NO❗ |
| └ | approve | External ❗ | 🛑  |NO❗ |
```

| └ | transferFrom | External ❗️ | 🛑 |NO❗️ |

||||||

| **IERC20Metadata** | Interface | IERC20 |||

| └ | name | External ❗️ | |NO❗️ |

| └ | symbol | External ❗️ | |NO❗️ |

| └ | decimals | External ❗️ | |NO❗️ |

||||||

| **ERC20** | Implementation | Context, IERC20, IERC20Metadata |||

| └ | <Constructor> | Public ❗️ | 🛑 |NO❗️ |

| └ | 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 🔒 | 🛑 | |

||||||

```
| **IUniswapV2Router01** | Interface |  |||
| └ | factory | External ❗️ |   |NO❗️ |
| └ | WETH | External ❗️ |   |NO❗️ |
| └ | addLiquidity | External ❗️ | 🛑  |NO❗️ |
| └ | addLiquidityETH | External ❗️ |  💵 |NO❗️ |
| └ | removeLiquidity | External ❗️ | 🛑  |NO❗️ |
| └ | removeLiquidityETH | External ❗️ | 🛑  |NO❗️ |
| └ | removeLiquidityWithPermit | External ❗️ | 🛑  |NO❗️ |
| └ | removeLiquidityETHWithPermit | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactTokensForTokens | External ❗️ | 🛑  |NO❗️ |
| └ | swapTokensForExactTokens | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactETHForTokens | External ❗️ |  💵 |NO❗️ |
| └ | swapTokensForExactETH | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactTokensForETH | External ❗️ | 🛑  |NO❗️ |
| └ | swapETHForExactTokens | External ❗️ |  💵 |NO❗️ |
| └ | quote | External ❗️ |   |NO❗️ |
| └ | getAmountOut | External ❗️ |   |NO❗️ |
| └ | getAmountIn | External ❗️ |   |NO❗️ |
| └ | getAmountsOut | External ❗️ |   |NO❗️ |
| └ | getAmountsIn | External ❗️ |   |NO❗️ |
||||||
| **IUniswapV2Router02** | Interface | IUniswapV2Router01 |||
| └ | removeLiquidityETHSupportingFeeOnTransferTokens | External ❗️ | 🛑  |NO❗️ |
| └ | removeLiquidityETHWithPermitSupportingFeeOnTransferTokens | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactETHForTokensSupportingFeeOnTransferTokens | External ❗️ |  💵 |NO❗️ |
| └ | swapExactTokensForETHSupportingFeeOnTransferTokens | External ❗️ | 🛑  |NO❗️ |
||||||
```

| **IUniswapV2Factory** | Interface | |||

| └ | feeTo | External ❗️ | |NO❗️ |

| └ | feeToSetter | External ❗️ | |NO❗️ |

| └ | getPair | External ❗️ | |NO❗️ |

| └ | allPairs | External ❗️ | |NO❗️ |

| └ | allPairsLength | External ❗️ | |NO❗️ |

| └ | createPair | External ❗️ | 🛑 |NO❗️ |

| └ | setFeeTo | External ❗️ | 🛑 |NO❗️ |

| └ | setFeeToSetter | External ❗️ | 🛑 |NO❗️ |

||||||

| **IWETH** | Interface | |||

| └ | deposit | External ❗️ | 💵 |NO❗️ |

| └ | withdraw | External ❗️ | 🛑 |NO❗️ |

||||||

| **AppleswapRouter** | Implementation | Ownable |||

| └ | <Constructor> | Public ❗️ | 🛑 |NO❗️ |

| └ | UNISWAP_V2_ROUTER | Internal 🔒 | | |

| └ | UNISWAP_FACTORY | Internal 🔒 | | |

| └ | setAmountGetFreeTax | Public ❗️ | 🛑 | onlyOwner |

| └ | setDexToken | Public ❗️ | 🛑 | onlyOwner |

| └ | setSwapFee | Public ❗️ | 🛑 | onlyOwner |

| └ | getSwapFee | Internal 🔒 | | |

| └ | swapExactTokensForTokens | Public ❗️ | 🛑 | validate |

| └ | swapTokensForExactTokens | Public ❗️ | 🛑 | validate |

| └ | swapExactTokensForETH | Public ❗️ | 🛑 | validate |

| └ | swapTokensForExactETH | Public ❗️ | 🛑 | validate |

| └ | swapExactETHForTokens | Public ❗️ | 💵 | validate |

| └ | swapETHForExactTokens | Public ❗️ | 💵 | validate |

| └ | getPair | External ❗ | |NO❗ |

| └ | getAmountIn | Public ❗ | |NO❗ |

| └ | getAmountOut | Public ❗ | |NO❗ |

| └ | _getWETHPath | Internal 🔒 | | |

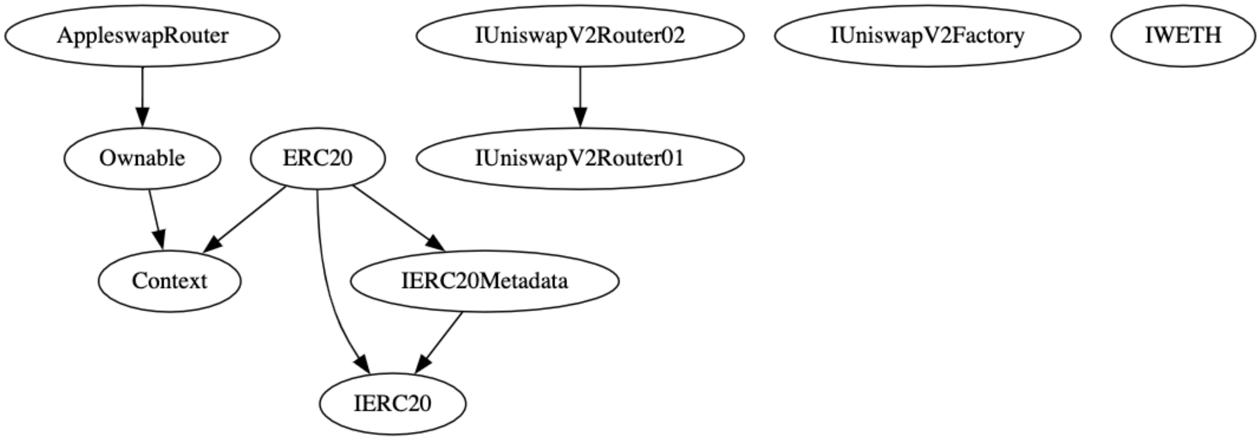
| └ | _transfer | Internal 🔒 | 🛑 | |

| └ | setTeamWallet | Public ❗ | 🛑 | onlyOwner |

| └ | _approve | Internal 🔒 | 🛑 | |

| └ | <Receive Ether> | External ❗ | 💵 |NO❗ |

INHERITANCE GRAPH



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized privileges	Major 🟠

`onlyOwner` centralized privileges are listed below:

```
renounceOwnership()
transferOwnership()
setAmountGetFreeTax()
setDexToken()
setSwapFee()
setTeamWallet()
```

RECOMMENDATION

Deployers, contract owners, administrators, access controlled, and all other privileged roles' private-keys/access-keys/admin-keys should be secured carefully. These entities can have a single point of failure that compromises the security of the project. Manage centralized and privileged roles carefully, review PAGE 09 for more information.

ACKNOWLEDGEMENT

Apple Swap team has argued that privileged roles are used as intended. Auditor recommended to:

Implement multi-signature wallets: Require multiple signatures from different parties to execute certain sensitive functions within contracts. This spreads control and reduces the risk of a single party having complete authority.

Use a decentralized governance model: Implement a governance model that enables token holders or other stakeholders to participate in decision-making processes. This can include voting on contract upgrades, parameter changes, or any other critical decisions that impact the contract's functioning.

Identifier	Definition	
ASR-01	Flash loan attack vector	

While smart contract itself doesn't have flash loan functions, it's susceptible to being attacked through a flash loan. An attacker can take a flash loan from another protocol, interact with smart contract, and exploit potential pricing discrepancies.

NOTE

Smart contract isn't inherently flawed, but its interactions with broader DeFi ecosystem can create vulnerabilities, and exploits.

Identifier	Definition	Severity
LOG-01	Lack of appropriate arbitrary boundaries	Minor

Below mentioned functions are set with inappropriate arbitrary boundaries.

```
setAmountGetFreeTax()
setSwapFee()
```


RECOMMENDATION

These functions should be provided appropriate upper and lower boundaries.

Identifier	Definition	Severity
LOG-02	Potential front-running	Minor

Potential front-running also classified as – sandwich attack happens when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by front-running a transaction to purchase assets and make profits by back-running a transaction to sell assets. Below mentioned functions are called without setting restrictions on slippage or minimum output:

```
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapTokensForExactTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
swapTokensForExactETH()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapETHForExactTokens()
```


RECOMMENDATION

These functions should be provided reasonable minimum output amounts, instead of zero.

Identifier	Definition	
LOG-03	Re-entrancy	

Vulnerability associated with re-entrancy is when Ether is sent out before the state is updated. Since Solidity `0.5.0`, re-entrancy due to receiving Ether is less likely because the fallback function is marked `payable`.

```
receive() external payable {}
```

Smart contract uses `transfer` method to move assets around, which in the provided methods are done after important state changes and should deter re-entrancy attacks.

Identifier	Definition	Severity
COD-02	Timestamp manipulation via `block.timestamp`	Minor

Be aware that the timestamp of the block can be manipulated by a miner. When the contract uses the timestamp to seed a random number, the miner can actually post a timestamp within 15 seconds of the block being validated, effectively allowing the miner to precompute an option more favorable to their chances.

RECOMMENDATION

To maintain block integrity, follow 15 seconds rule, and scale time dependent events accordingly.

Identifier	Definition	
COD-07	Conformance to solidity writing guide	

Use more intuitive variable names, e.g., `teamFee` may be named `teamFeeBasisPoints` since it ranges from `0 to 10000`.

RECOMMENDATION

Follow coding conventions for writing solidity code. Learn more:

https://docs.soliditylang.org/en/v0.8.16/style-guide.html

Identifier	Definition	Severity
COD-08	No handling for failed transfers	Minor

Smart contract assumes that all ERC20 transfers, e.g., `transferFrom` and `transfer`, will always succeed. Some ERC20 tokens `revert` on failure, while others return `false`.

RECOMMENDATION

Handle failed transfers accurately.

Identifier	Definition	Severity
COD-09	Lack of contract balance withdraw	

Smart contract may collect tokens, and ethers from external addresses. Some swap, and liquidity-add events may accumulate residual ethers, and tokens.

RECOMMENDATION

Add `withdraw()` function to take out tokens and ethers from the contract.

Identifier	Definition	Severity
COD-10	Direct and indirect dependencies	Unknown
ASR-02	External Calls with ETH	

Smart contract implementation is highly dependent on Uniswap's router implementation, such as:

```
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapTokensForExactTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
swapTokensForExactETH()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapETHForExactTokens()
_approve()
```

Smart contract is interacting with protocols e.g., External Contracts, Market Makers, Web 3 applications, Open Zeppelin tools. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

External Calls with ETH - Unwanted behaviors can be triggered by malicious contracts when transferring Ether.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary. Only interact with trusted contracts, libraries, and interfaces.

Identifier	Definition	Severity
COD-11	Arbitrary function call	Major
ASR-03	Gas token minting	

_approve function uses low-level call to make the approval, making it prone to vulnerabilities.

_approve function forces allowance of a token to be reset to zero before setting it to a new value. The method to reset allowance first could allow for a griefing vector where someone could potentially mint GasTokens. However, some tokens, e.g., USDT, may require this step. Be aware of the potential for GasToken minting.

RECOMMENDATION

Use the standard ERC20 approve function for safer execution.

Identifier	Definition	Severity
COD-12	Lack of event-driven architecture	Minor

Smart contract uses function calls to update state, which can make it difficult to track and analyze changes to the contract over time.

RECOMMENDATION

Use events to track state changes. Events improve transparency and provide a more granular view of contract activity.

Identifier	Definition	Severity
VOL-01	Irrelevant code	Minor

Redundant fee calculation in multiple functions. Calculate fee once and use that value in multiple functions instead.

RECOMMENDATION

Fix redundant code.

Identifier	Definition	
COM-01	Floating compiler status	

Compiler is set to `^0.8.0`

RECOMMENDATION

Pragma should be fixed to the version that you’re indenting to deploy your contracts with.

DISCLAIMERS

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. 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. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT INTERFI NETWORK

InterFi Network provides intelligent blockchain solutions. We provide solidity development, testing, and auditing services. We have developed 150+ solidity codes, audited 1000+ smart contracts, and analyzed 500,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Velas, Oasis, etc.

InterFi Network is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 6+ casual contributors.

Website: https://interfi.network

Email: hello@interfi.network

GitHub: https://github.com/interfinetwork

Telegram (Engineering): https://t.me/interfiaudits

Telegram (Onboarding): https://t.me/interfisupport

interfinetwork

hello@interfi.network

https://interfi.network

SMART CONTRACT AUDITS | SOLIDITY DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS