SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

AUSTRALIAN CRYPTO COIN GREEN

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	Australian Crypto Coin Green
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0xD436F4f4F309f3eeC38b33071CcC7115630a1F1C
Blockchain	Binance Smart Chain
Centralization	Active ownership
Commit	475ecb0de0ae3d8b526419097150224090b46be4
Website	https://accoin.com.au/
Telegram	https://t.me/Accoinaus/
Facebook	https://www.facebook.com/accoinaus/
Twitter	https://twitter.com/AccoinCrypto/
Report Date	September 07, 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	1	0	1	8	0
Acknowledged	1	0	1	1	1
Resolved	0	1	0	0	0
ACCG Property Token Important Privileges	**Mint, Finalize, Self-Destruct,** Withdraw Excess Eth, Withdraw Excess Token, Vest Tokens				
ACCG Token Important Privileges	**Destroy Property Token, Finalize Property, Pause Contract,** Withdraw Excess Eth, Withdraw Excess Token				

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 Australian Crypto Coin Green 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:

- ACCG.sol

If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Public Contract Link	
https://bscscan.com/address/0xD436F4f4F309f3eeC38b33071CcC7115630a1F1C#code	
Contract Name	ACCG
Compiler Version	0.8.18
License	MIT

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.

ℹ️ 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

```
v1.0 — automatedCheck — 09192023
| **SafeMath** | Library |  |||
| └ | tryAdd | Internal 🔒 |   | |
| └ | trySub | Internal 🔒 |   | |
| └ | tryMul | Internal 🔒 |   | |
| └ | tryDiv | Internal 🔒 |   | |
| └ | tryMod | Internal 🔒 |   | |
| └ | add | Internal 🔒 |   | |
| └ | sub | Internal 🔒 |   | |
| └ | mul | Internal 🔒 |   | |
| └ | div | Internal 🔒 |   | |
| └ | mod | Internal 🔒 |   | |
| └ | sub | Internal 🔒 |   | |
| └ | div | Internal 🔒 |   | |
| └ | mod | Internal 🔒 |   | |
||||||
| **Context** | Implementation |  |||
| └ | _msgSender | Internal 🔒 |   | |
```

| └ | _msgData | Internal 🔒 | | |

||||||

| **Pausable** | Implementation | Context |||

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

| └ | paused | Public ❗️ | |NO❗️ |

| └ | _requireNotPaused | Internal 🔒 | | |

| └ | _requirePaused | Internal 🔒 | | |

| └ | _pause | Internal 🔒 | 🛑 | whenNotPaused |

| └ | _unpause | Internal 🔒 | 🛑 | whenPaused |

||||||

| **Ownable** | Implementation | Context |||

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

| └ | owner | Public ❗️ | |NO❗️ |

| └ | _checkOwner | Internal 🔒 | | |

| └ | renounceOwnership | Public ❗️ | 🛑 | onlyOwner |

INTERFI CONFIDENTIAL

| └ | 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 🔒 | 🛑 | |

||||||

| **ACCGPropertyToken** | Implementation | ERC20, Ownable |||

| └ | <Constructor> | Public ❗ | 🛑 | ERC20 |

| └ | setNewManager | External ❗ | 🛑 | onlyManager |

| └ | totalVested | Public ❗ | |NO❗ |

| └ | vestedBalanceOf | Public ❗ | |NO❗ |

| └ | vestingStart | Public ❗ | |NO❗ |

| └ | vestTokens | Public ❗ | 🛑 | onlyManager |

| └ | claimVestedTokens | Public ❗ | 🛑 | nonReentrant |

| └ | mint | Public ❗ | 🛑 | onlyOwner |

| └ | burn | Public ❗ | 🛑 | onlyOwner |

| └ | withdrawExcessEth | Public ❗ | 🛑 | onlyOwner |

| └ | withdrawExcessToken | Public ❗ | 🛑 | onlyOwner |

| └ | finalize | Public ❗ | 🛑 | onlyOwner |

| └ | selfDestruct | Public ❗ | 🛑 | onlyOwner |

||||||

| **IPriceFeed** | Interface | |||

| └ | latestAnswer | External ❗ | |NO❗ |

| └ | decimals | External ❗ | |NO❗ |

||||||

| **IUniswapV2Router02** | Interface | |||

| └ | getAmountsOut | External ❗ | |NO❗ |

| └ | WETH9 | External ❗ | |NO❗ |

||||||

| **ACCG** | Implementation | Ownable, Pausable, ERC20 |||

| └ | <Constructor> | Public ❗ | 🛑 | ERC20 |

| └ | setBNBPriceFeed | External ❗ | 🛑 | onlyOwner |

| └ | setAccPriceFeed | External ❗ | 🛑 | onlyOwner |

| └ | setIsLivePriceOn | Public ❗ | 🛑 | onlyOwner |

| └ | createPropertyToken | Public ❗ | 🛑 | onlyOwner nonReentrant |

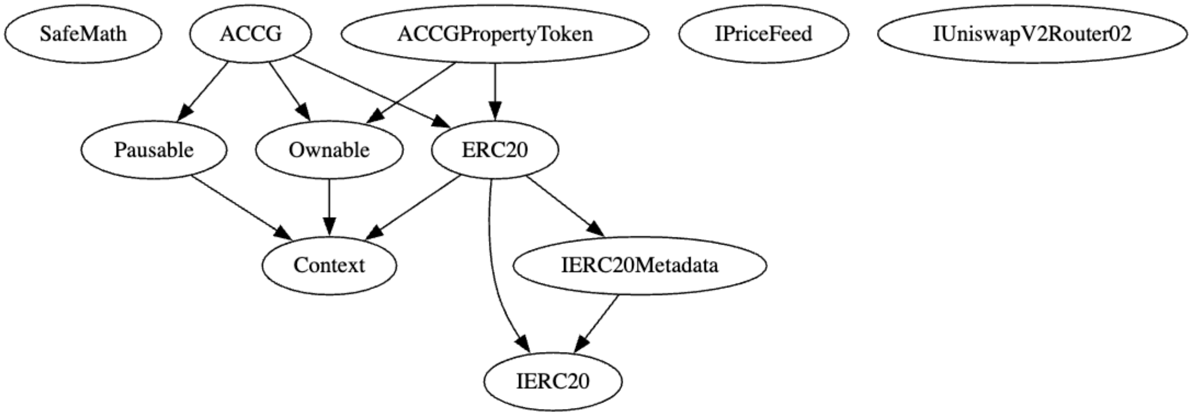
| └ | getACCLivePrice | Public ❗ | |NO❗ |

| └ | getBNBLivePrice | Public ❗ | |NO❗ |

| └ | getACCGPrice | Public ❗ | |NO❗ |

```
| └ | purchasePropertyToken | Public ❗ | 💵 | nonReentrant |
| └ | getACCPropertyTokenPrice | Public ❗ |   |NO❗ |
| └ | getETHPrice | Public ❗ |   |NO❗ |
| └ | pause | Public ❗ | 🛑 | onlyOwner |
| └ | unpause | Public ❗ | 🛑 | onlyOwner |
| └ | withdrawExcessEth | External ❗ | 🛑 | onlyOwner |
| └ | withdrawExcessToken | External ❗ | 🛑 | onlyOwner |
| └ | destroyPropertyToken | External ❗ | 🛑 | onlyOwner |
| └ | finalizeProperty | External ❗ | 🛑 | onlyOwner |
| └ | burnPropertyToken | Public ❗ | 🛑 | onlyOwner |
| └ | <Receive Ether> | External ❗ | 💵 |NO❗ |
```

INHERITANCE GRAPH



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized privileges	Critical 🔴
CEN-07	Authorizations and access control	
CEN-05	Privileged role pauses **`ACCG`** smart contract	
ACC-01	Privileged role calls `selfDestruct` to destroy contracts and withdraw funds in **`ACCGPropertyToken`** and **`ACCG`** contracts	

Important `onlyOwner` centralized privileges are listed below:

```
mint()
burn()
withdrawExcessEth()
withdrawExcessToken()
finalize()
selfDestruct()
setBNBPriceFeed()
setAccPriceFeed()
setIsLivePriceOn()
createPropertyToken()
pause()
unpause()
withdrawExcessEth()
withdrawExcessToken()
destroyPropertyToken()
finalizeProperty()
burnPropertyToken()
```

`onlyManager` access control is provided to:

```
setNewManager()
vestTokens()
```

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.

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.

ACKNOWLEDGEMENT

ACC team has argued that privileged roles are used as intended, and accepted to use multi-signature wallets to manage centralization wherever possible.

Identifier	Definition	Severity
ACC-03	Use of `selfDestruct` in **`ACCGPropertyToken`** and **`ACCG`** contracts	Critical 🔴

`selfDestruct()` destroys smart contracts and transfers all remaining balance to the provided address.

Below mentioned functions use `selfDestruct()` directly or indirectly:

```
finalize()
destroyPropertyToken()
finalizeProperty()
```


RECOMMENDATION

Remove `selfDestruct()` – as it deliberately destroys smart contracts.

Identifier	Definition	Severity
CEN-02	Initial asset distribution in **ACCGPropertyToken** and **ACCG** contracts	Minor
CEN-11	Asset mint in **ACCGPropertyToken** contract	

All of initially minted assets are sent to **msg.sender** when deploying contracts. This can be an issue as deployer can distribute tokens without consulting the community.

```
contract ACCGPropertyToken is ERC20, Ownable {
        _mint(msg.sender, initialSupply);


    function mint(address _receiver, uint256 _amount) public onlyOwner {
        _mint(_receiver, _amount);
    }


contract ACCG is Ownable, Pausable, ERC20 {
    constructor() ERC20("ACCG", "ACCG") {
        _mint(msg.sender, 1000000000000000 * 1e18);
```


CEN-02 RECOMMENDATION

Project must communicate with stakeholders and obtain the community consensus while distributing assets.

CEN-11 RECOMMENDATION

Declare and lock total asset supply. Access to mint function negatively elevates centralization risk.

Identifier	Definition	Severity
LOG-01	Lack of adequate checks in **`ACCGPropertyToken`** and **`ACCG`** contracts	Medium 🟡

Below mentioned functions should be provided adequate `require` checks:

`createPropertyToken()` – No limit on how much fees can be charged. Limit maximum fees to be charged.

`purchasePropertyToken()` – No access control, meaning anyone can call it. Use appropriate access control, e.g., `onlyAuthorized`.

`claimVestedTokens()` – No access control, meaning anyone can call it. Use appropriate access control, e.g., `onlyAuthorized`.

RECOMMENDATION

These functions should be provided appropriate adequate `require` checks.

ACKNOWLEDGEMENT

ACC team has argued that functions like `purchasePropertyToken()` and `claimVestedTokens()` should be callable by anyone as per logic design.

Identifier	Definition	Severity
LOG-02	Note regarding front-running	Minor

Smart contract doesn't have any explicit mechanisms like slippage control or transaction ordering protection to prevent front-running attacks. In token contract, this is a significant issue. Anyone watching the mempool can potentially front-run an operation to gain some advantage.

NOTE

Without explicit measures to control transaction ordering, miners or users may potentially manipulate the order of transactions for their benefit.

Identifier	Definition	Severity
LOG-03	Re-entrancy in **ACCGPropertyToken** and **ACCG** contracts	Major

Below mentioned function is used without re-entrancy guard:

```
claimVestedTokens()
purchasePropertyToken()
```


RECOMMENDATION

Use Checks Effects Interactions pattern when handing over the flow to an external entity and/or guard functions against re-entrancy attacks. Re-entrancy guard is used to prevent re-entrant calls.

RESOLUTION

ACC team has added re-entrancy guard to above mentioned functions.

Identifier	Definition	Severity
COD-02	Timestamp manipulation via `block.timestamp` in **`ACCGPropertyToken`** and **`ACCG`** contracts	Minor

Be aware that the timestamp of the block can be manipulated by a miner. Smart contracts rely on `block.timestamp` for determining when vest started and claimed. Miners have some degree of control over this value, which could lead to minor manipulations.

RECOMMENDATION

To maintain block integrity, scale time dependent events accordingly.

Identifier	Definition	Severity
COD-06	Unknown accounts and contracts in **ACCG** contract	Minor

Below mentioned `0x` are found in smart contracts:

```
address public bnbPriceFeed = 0xd0D5e3DB44DE05E9F294BB0a3bEEaF030DE24Ada;
address public accPriceFeed = 0x7d7356bF6Ee5CDeC22B216581E48eCC700D0497A;
```


RECOMMENDATION

Dependencies on malicious externally owned accounts and contracts can introduce vulnerabilities. Only interact with trusted accounts and contracts.

Identifier	Definition	Severity
COD-10	Direct and indirect dependencies in **ACCGPropertyToken** and **ACCG** contracts	Unknown

Smart contracts are interacting with third party protocols e.g., Price Feeds, Market Makers, Base Token, Web 3 Applications, Uniswap and Open Zeppelin tools. The scope of the audit treats these entities as black boxes and assumes their functional correctness. However, in the real world, all of them can be compromised, and exploited. Moreover, upgrades in these entities can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary.

ACKNOWLEDGEMENT

ACC team will inspect third party dependencies regularly, and push updates as required.

Identifier	Definition	
COD-13	Note regarding flash loan vulnerabilities	

Smart contract does not interact with external contracts for token swapping or lending, so it is less susceptible to flash loan attacks, which usually exploit some form of arbitrage opportunity. However, if smart contract interacts with malicious contract, technically flash loan vulnerabilities can be introduced.

NOTE

Due to the interconnected nature of DeFi contracts, smart contracts can be exploited using another malicious contract. Be cautious while interacting with third-party contracts, tokens, and protocols.

Identifier	Definition	Severity
COD-12	Lack of event-driven architecture in **ACCGPropertyToken** and **ACCG** contracts	Minor

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

ACCGPropertyToken

```
setNewManager()
mint()
burn()
withdrawExcessEth()
withdrawExcessToken()
finalize()
selfDestruct()
```


ACCG

```
setIsLivePriceOn()
purchasePropertyToken()
pause()
unpause()
withdrawExcessEth()
withdrawExcessToken()
destroyPropertyToken()
finalizeProperty()
burnPropertyToken()
```

RECOMMENDATION

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

Identifier	Definition	Severity
VOL-01	Unchecked return values in **ACCGPropertyToken** and **ACCG** contracts	Minor

Calls `_token.transfer` and `transfer` are used without checking their return values, which should be `true` for a successful transaction.

RECOMMENDATION

Check return values to validate if calls are successful or not.

Identifier	Definition	Severity
VOL-02	Irrelevant code in **ACCGPropertyToken** contract	Minor

Redundant code in:

```
contract ACCGPropertyToken
```


RECOMMENDATION

Smart contract **ACCGPropertyToken** is in the code, but is not properly deployed. Remove redundant code, or use interface instead of the contract.

Identifier	Definition	Severity
VOL-03	Volatile code in **`ACCGPropertyToken`** and **`ACCG`** contracts	Medium 🟡

`ACCGPropertyToken`

- There is no way to update `baseToken`, making it immutable once set. If the baseToken is compromised or deprecated, it can't be changed.
- `_transfer` function doesn't check whether the contract has enough tokens to transfer. This could result in a failed transaction.

`ACCG`

- `createPropertyToken()` has a large number of parameters and local variables, which could cause 'Stack too deep' error.
- In `purchasePropertyToken`, there's no validation to check whether `propertyId` exists or not.
- Multiplying `latestAnswer()` by `1e18` and then dividing by 10 times `IPriceFeed(accPriceFeed).decimals` logic is unclear, especially when decimals of two different price feeds are being used interchangeably.

RECOMMENDATION

Fix non-conforming logic.

Identifier	Definition	Severity
VOL-04	Possible integer underflow in **`ACCGPropertyToken`** and **`ACCG`** contracts	Minor

- In `vestTokens` function, `_totalVested = _totalVested - tokens;` can lead to underflow if `_totalVested` is less than `tokens`.
- In `burnPropertyToken` function, amount to be burned is not checked against the user's balance, which may lead to underflows.

RECOMMENDATION

Make sure above calculations does not cause underflow.

Identifier	Definition	Severity
COM-04	Unbounded loops in **`ACCG`** contract	Minor

`ACCG`

`getACCGPrice()` calls itself recursively when `isLivePriceOn` is `false`, leading to infinite recursion and out-of-gas error.

RECOMMENDATION

Fix infinite recursion in `getACCGPrice()`.

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