

CHENINJA AUDITS



Security Assessment

TFI Staking

October 26, 2022



Table of Contents

1 Assessment Summary

2 Technical Findings Summary

3 Project Overview

3.1 Token Summary

3.2 Risk Analysis Summary

3.3 Main Contract Assessed

4 Smart Contract Risk Checks

4.1 Mint Check

4.2 Fees Check

4.3 Blacklist Check

4.4 MaxTx Check

4.5 Pause Trade Check

5 Contract Ownership

6 Liquidity Ownership

7 KYC Check

8 Smart Contract Vulnerability Checks

8.1 Smart Contract Vulnerability Details

8.2 Smart Contract Inheritance Details

8.3 Smart Contract Privileged Functions

9 Assessment Results and Notes(Important)

10 Social Media Check(Informational)

11 Technical Findings Details

12 Disclaimer



Assessment Summary

This report has been prepared for TFI Staking on the Binance Smart Chain network. CFGNINJA provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.






The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.








Technical Findings Summary

Classification of Risk

Severity	Description
 Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
 Major	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
 Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
 Minor	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
 Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
 Critical	0	0	0
 Major	2	2	0
 Medium	0	0	0
 Minor	2	2	0
 Informational	1	1	0
Total	5	5	0



Project Overview

Token Summary

Parameter	Result
Address	
Name	TFI
Token Tracker	TFI (TFI-S)
Decimals	
Supply	
Platform	Binance Smart Chain
compiler	v0.8.17+commit.8df45f5f
Contract Name	STAKING
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/#code
Payment Tx	



Main Contract Assessed Contract Name

Name	Contract	Live
TFI		No

TestNet Contract Assessed Contract Name

Name	Contract	Live
TFI	00x533fBE9a5E485042D969d362FFa6693Add8e7A33	No

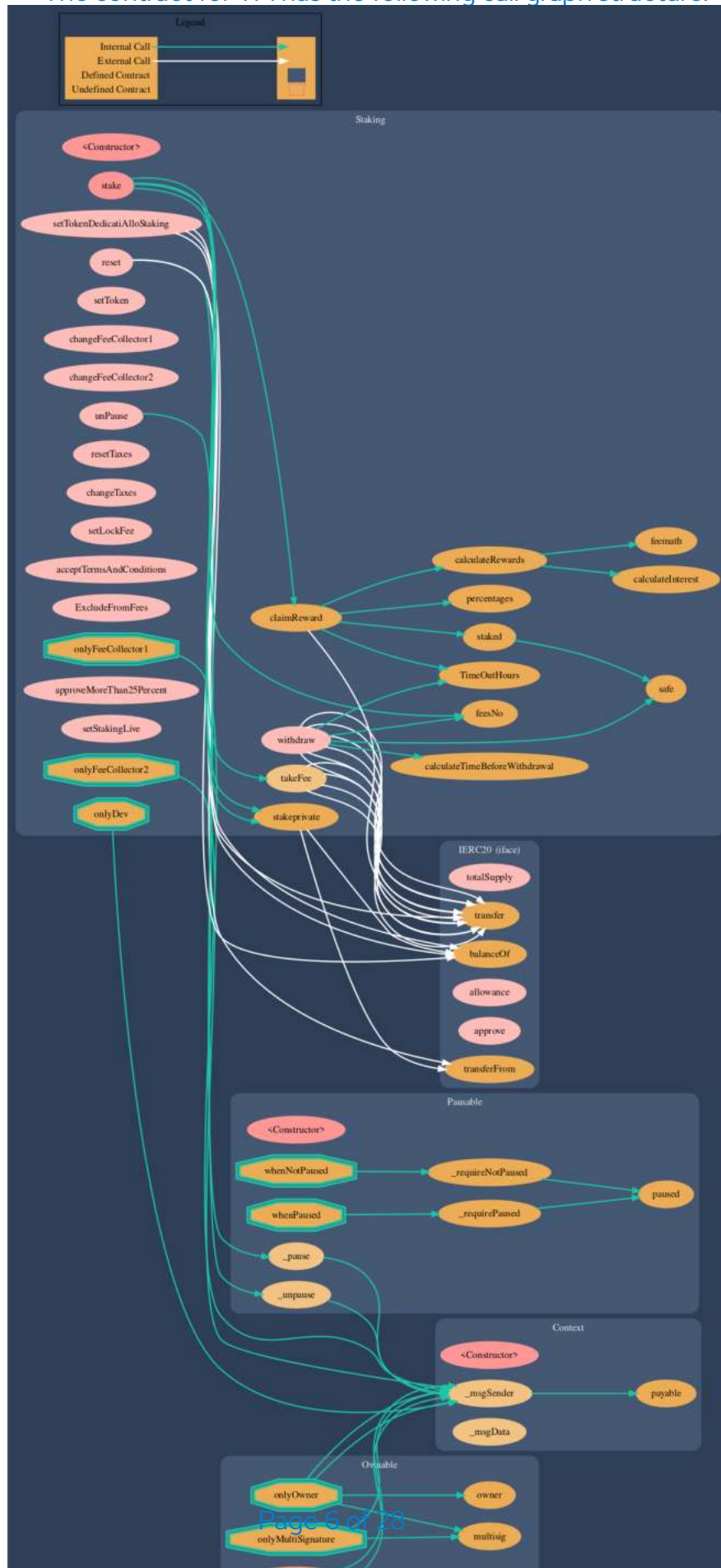
Solidity Code Provided

SolID	File Sha-1	FileName
staking	a266d6060817f0f591e8b6c0fa6634bf4bf64418	staking.sol



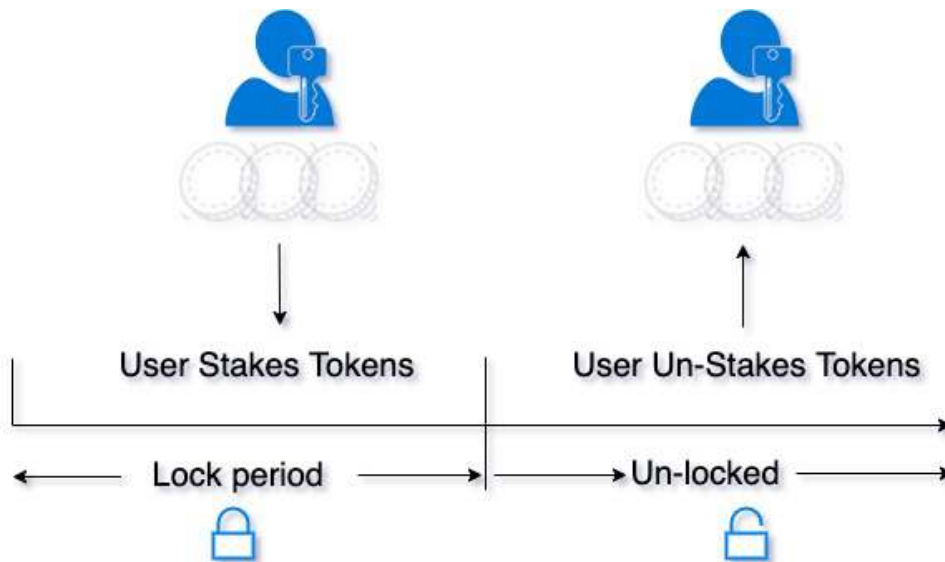
Call Graph

The contract for TFI has the following call graph structure.



What is a Staking Contract

A smart contract which allows users to stake and un-stake a specified ERC20 token. Staked tokens are locked for a specific length of time (set by the contract owner at the outset). Once the time period has elapsed, the user can remove their tokens again.



Reentrancy Check

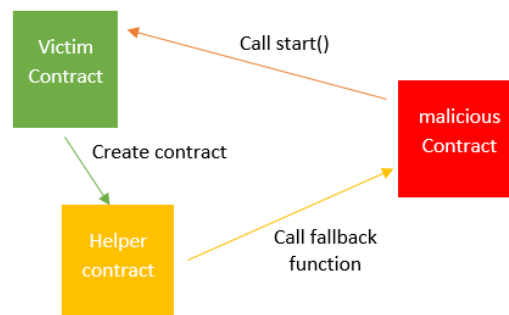
The Project Owners of TFI have not configured the Reentrancy Guard library.

You can read more about Reentrancy issues in the following link.

[Reentrancy After Istanbul.](#)

We recommend the team to add the library to the contract to avoid potential issues.

We recommend the team to create a new contract with Reentrancy Guard added to the same.



KYC Information

The Project Owners of TFI is not KYC.

KYC Information Notes:

Auditor Notes: No KYC Provided

Project Owner Notes:



Smart Contract Vulnerability Checks

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	staking.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	staking.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	staking.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	staking.sol	L: 3 C: 0
SWC-104	Pass	Unchecked Call Return Value.	staking.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	staking.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	staking.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	staking.sol	L: 0 C: 0
SWC-108	Pass	State variable visibility is not set..	staking.sol	L: 101 C: 30,L: 152 C: 9
SWC-109	Pass	Uninitialized Storage Pointer.	staking.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	staking.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	staking.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	staking.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-113	Pass	Multiple calls are executed in the same transaction.	staking.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	staking.sol	L: 0 C: 0
SWC-115	Low	Authorization through tx.origin.	staking.sol	L: 455 C: 15
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	staking.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	staking.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	staking.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	staking.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randomness.	staking.sol	L: 582 C: 49
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	staking.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	staking.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	staking.sol	L: 280 C: 31,L: 142 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	staking.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	staking.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	staking.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	staking.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	staking.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	staking.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	staking.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	staking.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	staking.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	staking.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	staking.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	staking.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	staking.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.



Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.



Smart Contract Vulnerability Details

SWC-115 - Authorization through tx.origin

CWE-477: Use of Obsolete Function

Description:

tx.origin is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract. A call could be made to the vulnerable contract that passes the authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.

Remediation:

tx.origin should not be used for authorization. Use msg.sender instead.

References:

Solidity Documentation - tx.origin

Ethereum Smart Contract Best Practices - Avoid using tx.origin

SigmaPrime - Visibility.



Smart Contract Vulnerability Details

SWC-120 – Weak Sources of Randomness from Chain Attributes

CWE-330: Use of Insufficiently Random Values

Description:

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B that also has a state variable x defined. This would result in two separate versions of x, one of them being accessed from contract A and the other one from contract B. In more complex contract systems this condition could go unnoticed and subsequently lead to security issues.

Shadowing state variables can also occur within a single contract when there are multiple definitions on the contract and function level.

Remediation:

Using commitment scheme, e.g. RANDAO. Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles. Using Bitcoin block hashes, as they are more expensive to mine.

References:

How can I securely generate a random number in my smart contract?)

When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.



Smart Contract Vulnerability Details

SWC-123 - Requirement Violation

CWE-573: Improper Following of Specification by Caller

Description:

The Solidity `require()` construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations. Violations of a requirement can indicate one of two possible issues:

- A bug exists in the contract that provided the external input.
- The condition used to express the requirement is too strong.

Remediation:

If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

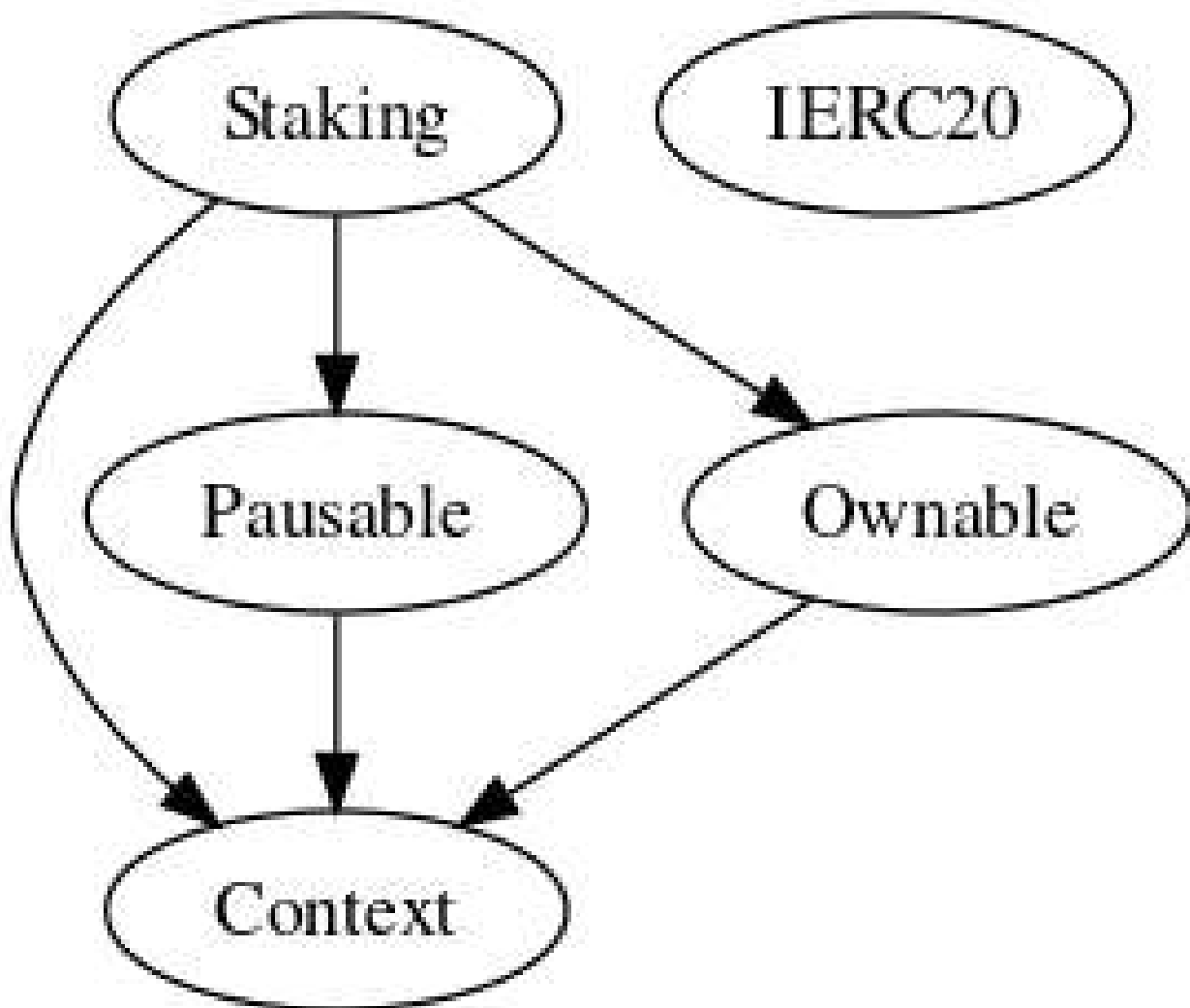
References:

The use of `revert()`, `assert()`, and `require()` in Solidity, and the new `REVERT` opcode in the EVM



Inheritance

The contract for TFI has the following inheritance structure.



Privileged Functions (onlyOwner)

Function Name	Parameters	Visibility
reset		external
setStakingLive		external
approveMoreThan25 Percent		external
setTokenDedicatiAllo Staking		external
ExcludeFromFees		external
changeTaxes		external
resetTaxes		external
unPause		external



Assessment Results

- The following is a staking contract, the dAPP has not been reviewed or audited by us during this assessment.
- There are a few coding improvements needed in the following code including the need for a nonreentrance and other minor areas. We suggest the dev team review those accordingly.
- Security Vulnerabilities have not been found and no major issues during the advance check and testing of the contract.
- No high-risk Exploits/Vulnerabilities Were Found in the Source Code.

Audit Passed





Smart Contract Advance Checks

ID	Severity	Name	Result	Status
TFI-S-01	Minor	Potential Sandwich Attacks.	Pass	In Progress
TFI-S-02	Informational	Function Visibility Optimization	Fail	Acknowledged
TFI-S-03	Minor	Lack of Input Validation.	Fail;	Pending
TFI-S-04	Major	Centralized Risk In addLiquidity.	Pass	Resolved
TFI-S-05	Major	Missing Event Emission.	Fail	Pending
TFI-S-06	Minor	Conformance with Solidity Naming Conventions.	Fail	Pending
TFI-S-07	Major	State Variables could be Declared Constant.	Fail	Pending
TFI-S-08	Major	Dead Code Elimination.	Pass	Pending
TFI-S-09	Major	Third Party Dependencies.	Pass	Pending
TFI-S-10	Major	Initial Token Distribution.	Pass	Pending
TFI-S-11	Major	Modification of Math Library	Pass	Pending
TFI-S-12	Major	Centralization Risks In The X Role	Pass	Pending
TFI-S-13	Informational	Extra Gas Cost For User..	Pass	Pending

During our assessment review, we manually check against the following test cases, this help us identify potential missing items from any automated tools. Is an additional safety measure to ensure the contract is safe and follow industry standards.



TFI-S-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Gas Optimization	 Informational	staking.sol: 329,19	 Acknowledged

Description

The following functions are declared as public and are not invoked in any of the contracts contained within the projects scope:

Function Name	Parameters	Visibility
stake		public

The functions that are never called internally within the contract should have external visibility

Remediation



We advise that the function's visibility specifiers are set to external, and the array-based arguments change their data location from memory to calldata, optimizing the gas cost of the function.

References:

external vs public best practices.



TFI-S-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	 Minor	staking.sol: 279,17,273,14,226,14	 Pending

Description

The given input is missing the check for the non-zero address.



Remediation

We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
...  
    require(receiver != address(0), "Receiver is the zero address");  
...
```



TFI-S-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	 Major	staking.sol: 230,14	 Pending

Description



Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes. The linked code does not create an event for the transfer.

Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.



TFI-S-06 | Conformance with Solidity Naming Conventions.

Category	Severity	Location	Status
Coding Style	 Minor	staking.sol: 432,14	 Pending

Description

Solidity defines a naming convention that should be followed. Rule exceptions: Allow constant variable name/symbol/decimals to be lowercase. Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
setTokenDedicatiAlloStaking
approveMoreThan25Percent
stakeprivate
feemath
```



Remediation

Follow the Solidity naming convention.

<https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-convention>



TFI-S-07 | State Variables could be Declared Constant.

Category	Severity	Location	Status
Coding Style	 Major	staking.sol: 179,20	 Pending

Description

Constant state variables should be declared constant to save gas.

Dev
ProjectOwne

Remediation

Add the constant attribute to state variables that never changes.

<https://docs.soliditylang.org/en/latest/contracts.html#constant-state-variables>



Social Media Checks

Social Media	URL	Result
Twitter		Fail
Other		Fail
Website	https://tfigroup.ae/tfi-token/	Pass
Telegram	https://t.me/tfitoken_official	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:



Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria and is readable by all the developers.



Disclaimer

CFGNINJA has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocacy for the Project, and users relying on this 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 the Project in question may pursue, and 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 entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

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. Information is provided 'as is, and CFGNINJA is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will CFGNINJA 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 or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA are subject to dependencies and are under continuing development. You agree that your access 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. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

