



CFG NINJA AUDITS

Security Assessment

ETERNAL2.0 Token

July 14, 2023

Audit Status: Fail

Audit Edition: Advanced



POWERED BY
BLADE POOL

Risk Analysis

Classifications of Manual Risk Results

Classification	Description
 Critical	Danger or Potential Problems.
 High	Be Careful or Fail test.
 Low	Pass, Not-Detected or Safe Item.
 Informational	Function Detected

Manual Code Review Risk Results

Contract Privilege	Description
 Buy Tax	1%
 Sale Tax	2%
 Cannot Sale	Pass
 Cannot Sale	Pass
 Max Tax	100%
 Modify Tax	Yes
 Fee Check	Fail
 Is Honeygot?	Detected
 Trading Cooldown	Not Detected
 Can Pause Trade?	Detected.



Contract Priviledge	Description
● Pause Transfer?	Detected
● Max Tx?	Fail
● Is Anti Whale?	Detected
● Is Anti Bot?	Not Detected
● Is Blacklist?	Detected
● Blacklist Check	Fail
● is Whitelist?	Detected
● Can Mint?	Pass
● Is Proxy?	Not Detected
● Can Take Ownership?	Not Detected
● Hidden Owner?	Not Detected
● Owner	0x02eece51fb80f5c3f5a318bd4aa0acf3a8a82f3b
● Self Destruct?	Not Detected
● External Call?	Not Detected
● Other?	Not Detected
● Holders	1
● Auditor Confidence	None

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.



Project Overview

Token Summary

Parameter	Result
Address	0x32f2166763932519604434874662D64c08c61Da2
Name	ETERNAL2.0
Token Tracker	ETERNAL2.0 (BETERNAL2.0)
Decimals	18
Supply	100,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.19+commit.7dd6d404
Contract Name	ETERNAL2
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x32f2166763932519604434874662D64c08c61Da2#code
Payment Tx	Corporate



Main Contract Assessed Contract Name

Name	Contract	Live
ETERNAL2.0	0x32f2166763932519604434874662D64c08c61Da2	Yes

TestNet Contract Assessed Contract Name

Name	Contract	Live
ETERNAL2.0	0xC433E435e7c6925e6C38C6047c8637DA548ef0De	Yes

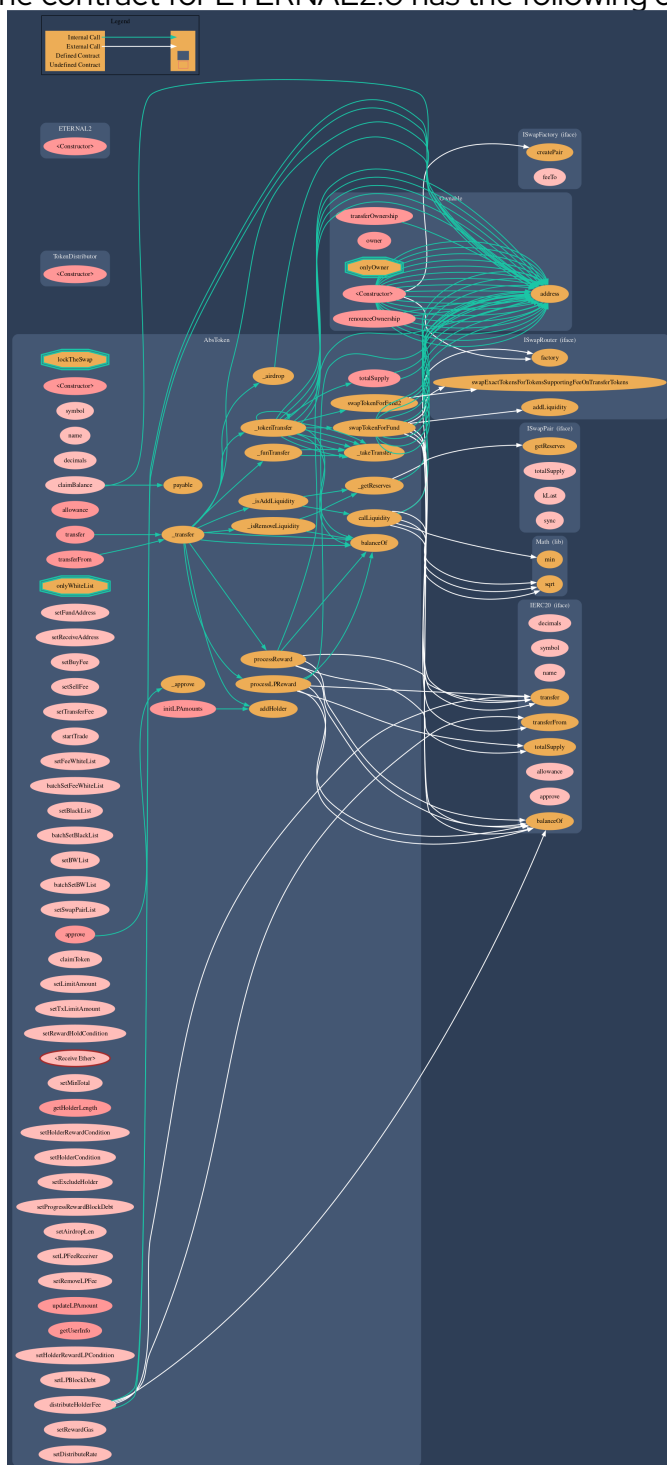
Solidity Code Provided

SolID	File Sha-1	FileName
Ethernals2	a12d8a5277a612f308e064a3385e2e858fd4b1a3	ethernals2.sol
Ethernals2		
Ethernals2		
Ethernals2		



Call Graph

The contract for ETERNAL2.0 has the following call graph structure.



Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	ethernals2.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	ethernals2.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	ethernals2.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	ethernals2.sol	L: 7 C: 0
SWC-104	Pass	Unchecked Call Return Value.	ethernals2.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	ethernals2.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	ethernals2.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	ethernals2.sol	L: 0 C: 0
SWC-108	Pass	State variable visibility is not set..	ethernals2.sol	L: 0 C: 0
SWC-109	Medium	Uninitialized Storage Pointer.	ethernals2.sol	L: 912 C: 12
SWC-110	Pass	Assert Violation.	ethernals2.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-111	Pass	Use of Deprecated Solidity Functions.	ethernals2.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	ethernals2.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	ethernals2.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	ethernals2.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	ethernals2.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	ethernals2.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	ethernals2.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	ethernals2.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	ethernals2.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randonmness.	ethernals2.sol	L: 367 C: 38
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	ethernals2.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	ethernals2.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	ethernals2.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	ethernals2.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	ethernals2.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-126	Pass	Insufficient Gas Griefing.	ethernals2.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	ethernals2.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	ethernals2.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	ethernals2.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	ethernals2.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	ethernals2.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	ethernals2.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	ethernals2.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	ethernals2.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	ethernals2.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	ethernals2.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-109 - Uninitialized Storage Pointer

CWE-824: Access of Uninitialized Pointer

Description:

Uninitialized local storage variables can point to unexpected storage locations in the contract, which can lead to intentional or unintentional vulnerabilities.

Remediation:

Check if the contract requires a storage object as in many situations this is actually not the case. If a local variable is sufficient, mark the storage location of the variable explicitly with the memory attribute. If a storage variable is needed then initialise it upon declaration and additionally specify the storage location storage. Note: As of compiler version 0.5.0 and higher this issue has been systematically resolved as contracts with uninitialised storage pointers do no longer compile.

References:

SigmaPrime - Uninitialised Storage Pointers



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?)

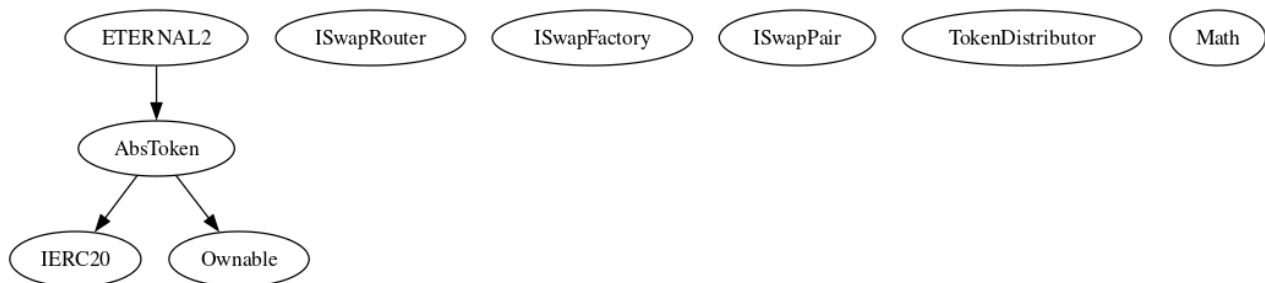
When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.



Inheritance

The contract for ETERNAL2.0 has the following inheritance structure.



Smart Contract Advance Checks

ID	Severity	Name	Result	Status
BETERNAL2.0-01	Low	Potential Sandwich Attacks.	Fail	Detected
BETERNAL2.0-02	Informational	Function Visibility Optimization	Pass	Detected
BETERNAL2.0-03	Low	Lack of Input Validation.	Fail	Detected
BETERNAL2.0-04	High	Centralized Risk In addLiquidity.	Fail	Detected
BETERNAL2.0-05	Low	Missing Event Emission.	Fail	Detected
BETERNAL2.0-06	Low	Conformance with Solidity Naming Conventions.	Pass	Not-Found
BETERNAL2.0-07	Low	State Variables could be Declared Constant.	Pass	Not-Found
BETERNAL2.0-08	Low	Dead Code Elimination.	Pass	Not-Found
BETERNAL2.0-09	High	Third Party Dependencies.	Pass	Detected
BETERNAL2.0-10	High	Initial Token Distribution.	Pass	Not-Found
BETERNAL2.0-11	High	claimStuckTokens can claim own tokens.	Pass	Detected
BETERNAL2.0-12	High	Centralization Risks In The X Role	Pass	Not-Found
BETERNAL2.0-13	Informational	Extra Gas Cost For User..	Fail	Detected



ID	Severity	Name	Result	Status
BETERNAL2.0-14	Medium	Unnecessary Use Of SafeMath	Pass	Not Detected
BETERNAL2.0-15	Medium	Symbol Length Limitation due to Solidity Naming Standards.	Pass	Detected
BETERNAL2.0-16	Medium	Taxes can be up to 100%	Pass	Not Detected
BETERNAL2.0-17	Logical Issue	Highly Permissive Role Access,`	Pass	Detected
BETERNAL2.0-18	Critical	Stop Transactions by using Enable Trade.	Fail	Detected



BETERNAL2.0-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	 Low	ethernals2.sol: L: 609, C: 14	 Detected

Description

A sandwich attack might happen 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 frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()
- addLiquidityETH()

Remediation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

References:

What Are Sandwich Attacks in DeFi – and How Can You Avoid Them?.



BETERNAL2.0-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	 Low	ethernals2.sol: L: 787 C: 14, L: 289 C: 14	 Detected

Description

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

The given input is missing the check for the allOnly Owners.

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");  
...  
...  
require(value X limitation, "Your not able to do this function");  
...
```

We also recommend customer to review the following function that is missing a required validation. allOnly Owners.



BETERNAL2.0-04 | Centralized Risk In addLiquidity.

Category	Severity	Location	Status
Coding Style	 High	ethernals2.sol: L:636, C: 44	 Detected

Description

`uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this), tokenAmount, 0, 0, owner(), block.timestamp);`

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function with the `to` address specified as `owner()` for acquiring the generated LP tokens from the BETERNAL2.0-WBNB pool.

As a result, over time the `_owner` address will accumulate a significant portion of LP tokens. If the `_owner` is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Remediation

We advise the `to` address of the `uniswapV2Router.addLiquidityETH` function call to be replaced by the contract itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

1. Indicatively, here are some feasible solutions that would also mitigate the potential risk:

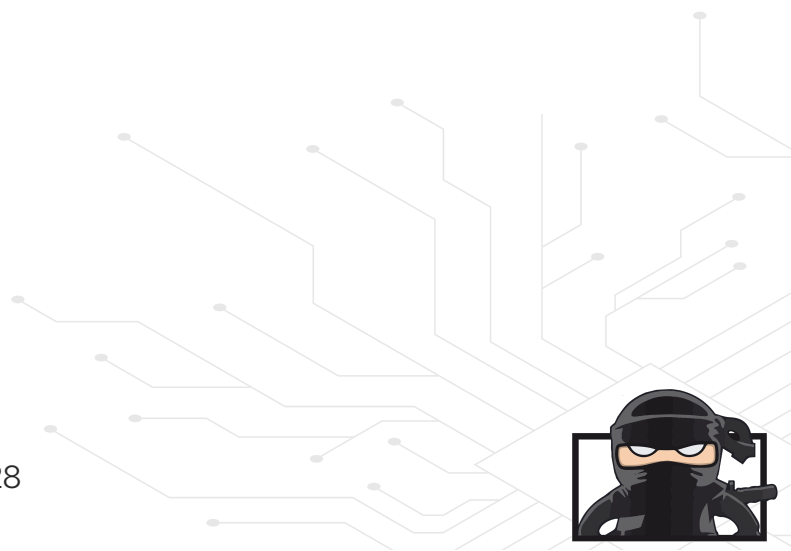
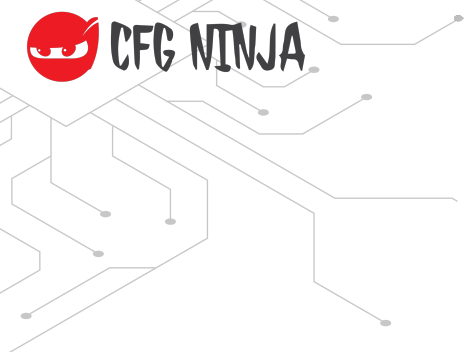
2. Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;

3. Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;

Introduction of a DAO / governance / voting module to increase transparency and user involvement

Project Action





BETERNAL2.0-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	 Low	ethernals2.sol: L: 557 C: 14	 Detected

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.



BETERNAL2.0-13 | Extra Gas Cost For User.

Category	Severity	Location	Status
Logical Issue	 Informational	ethernals2.sol: L: 702, C: 0	 Detected

Description

The user may trigger a tax distribution during the transfer process, which will cost a lot of gas and it is unfair to let a single user bear it.

Remediation

We advise the client to make the owner responsible for the gas costs of the tax distribution.

Project Action



BETERNAL2.0-18 | Stop Transactions by using Enable Trade.

Category	Severity	Location	Status
Logical Issue	 Critical	ethernals2.sol: L: 716 C: 0	 Detected

Description

Enable Trade is present on the following contract and when combined with Exclude from fees it can be considered a whitelist process, this will allow anyone to trade before others and can represent and issue for the holders.

Remediation

We recommend the project owner to carefully review this function and avoid problems when performing both actions.

Project Action



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.
 High	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
 Low	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	1	0	0
 High	1	0	0
 Medium	1	0	0
 Low	2	0	0
 Informational	1	0	0
Total	6	0	0



Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/LorraineN50473	Pass
Other		Fail
Website	https://eternal20.co/	Pass
Telegram	https://t.me/ete00000	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:



Assessment Results

Score Results

Review	Score
Overall Score	58/100
Auditor Score	50/100
Review by Section	Score
Manual Scan Score	1/33
SWC Scan Score	31/37
Advance Check Score	26/30

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 80 Points, if a project does not attain 80% is an automatic failure. Read our notes and final assessment below.

Audit Fail



Assessment Results

Important Notes:

- One vulnerability was found that needs to be addressed.
- Blacklist Present, Enable Trade Present and taxes to 100%.
- Please DYOR on the project.

Auditor Score =50
Audit Fail



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.

