CFG NINJA AUDITS

Security Assessment

Mysterious Giant Dragon Token

June 29, 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.
ℹ Informational	Function Detected

Manual Code Review Risk Results

Contract Priviledge	Description
🟢 Buy Tax	9%
🟢 Sale Tax	10%
🟢 Cannot Sale	Pass
🟢 Cannot Sale	Pass
🟢 Max Tax	25%
ℹ Modify Tax	Yes
🟢 Fee Check	Pass
🟢 Is Honeypot?	Detected, unable to buy and sale.
🟢 Trading Cooldown	Not Detected
🔴 Can Pause Trade?	Detected, Owner need to enable trade.

Contract Priviledge	Description
🔴 Pause Transfer?	Detected, Owner needs to enable trading.
🟢 Max Tx?	Fail
🟢 Is Anti Whale?	Detected
🟢 Is Anti Bot?	Not Detected
🟠 Is Blacklist?	Detected
🟢 Blacklist Check	Pass
🟢 is Whitelist?	Not Detected
🟢 Can Mint?	Pass
🟢 Is Proxy?	Not Detected
🟢 Can Take Ownership?	Not Detected
🟢 Hidden Owner?	Not Detected
ℹ Owner	0x7c652aef9462b1bbcd4a34b090eeba7c17cd3288
🟢 Self Destruct?	Not Detected
🟢 External Call?	Not Detected
ℹ Other?	Detected
🟢 Holders	1
🟢 Auditor Confidence	Low

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	0x4820d82B74E8A1cCe0C0BCC698B9fbC272588888
Name	Mysterious Giant Dragon
Token Tracker	Mysterious Giant Dragon (MGD)
Decimals	18
Supply	1,000,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.4+commit.c7e474f2
Contract Name	FatToken
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x4820d82B74E8A1cCe0C0BCC698B9fbC272588888#code
Payment Tx	Corporate

Main Contract Assessed
Contract Name

Name	Contract	Live
Mysterious Giant Dragon	0x4820d82B74E8A1cCe0C0BCC698B9fbC272588888	Yes

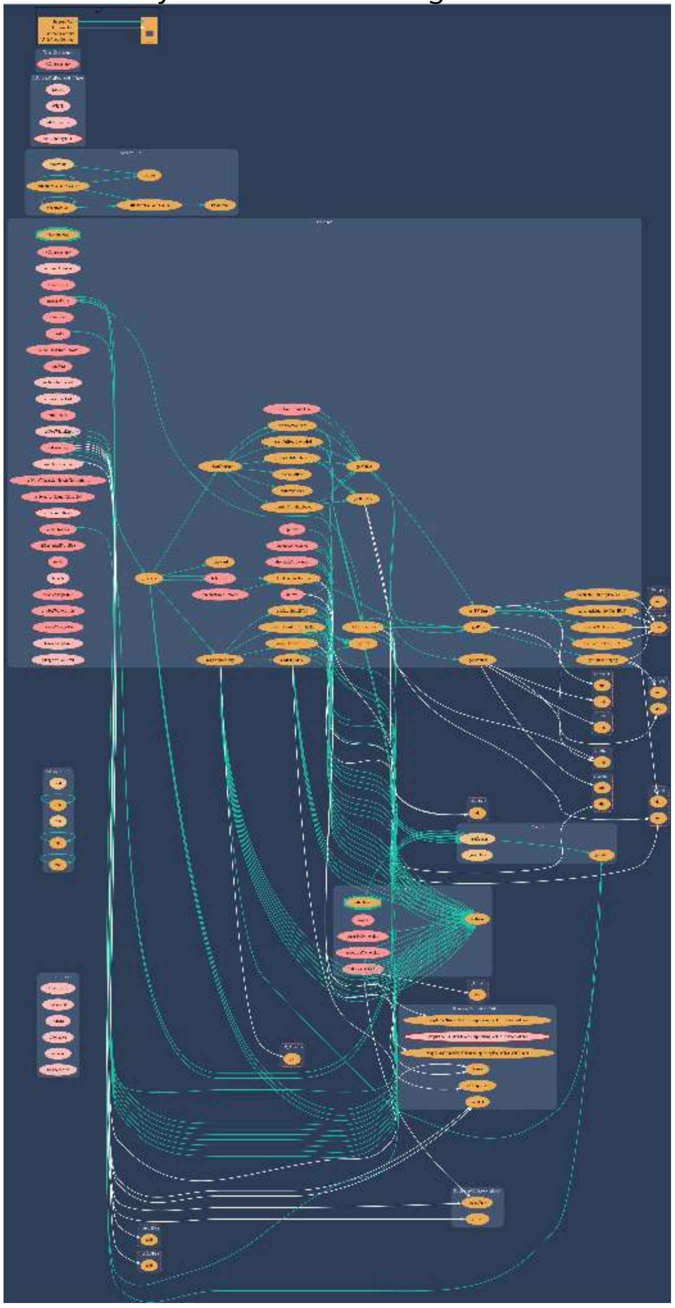
TestNet Contract was Not Assessed

Solidity Code Provided

SolID	File Sha-1	FileName
InSpace	cca460491ba6cc6179324b755a211119a65bd119	6-15-2023 InSpace.sol
InSpace		
InSpace		
InSpace		

Call Graph

The contract for Mysterious Giant Dragon 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	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-103	Fail	A floating pragma is set.	6-15-2023 InSpace.sol	L: 12 C: 0
SWC-104	Pass	Unchecked Call Return Value.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-108	Low	State variable visibility is not set..	6-15-2023 InSpace.sol	L: 630 C: 12
SWC-109	Pass	Uninitialized Storage Pointer.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	6-15-2023 InSpace.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-111	Pass	Use of Deprecated Solidity Functions.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-115	Low	Authorization through tx.origin.	6-15-2023 InSpace.sol	L: 787 C: 22
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randonmness.	6-15-2023 InSpace.sol	L: 1267 C: 26
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	6-15-2023 InSpace.sol	L: 680 C: 11
SWC-124	Pass	Write to Arbitrary Storage Location.	6-15-2023 InSpace.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-125	Pass	Incorrect Inheritance Order.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	6-15-2023 InSpace.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	6-15-2023 InSpace.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-108 - State Variable Default Visibility

CWE-710: Improper Adherence to Coding Standards

Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables

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:

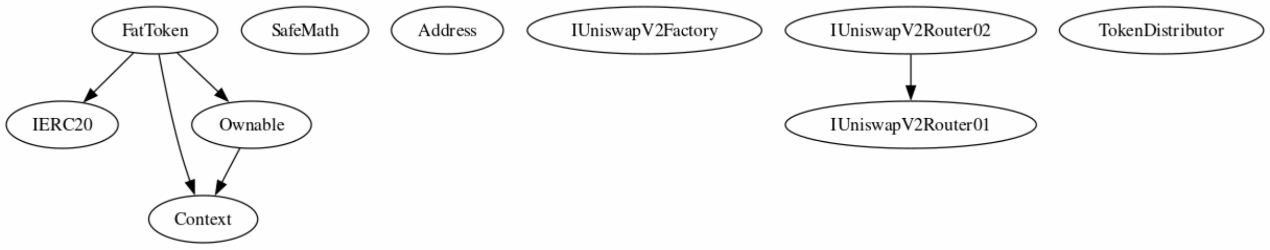
If the required logical condition is too strong, it should be weakened to allow all valid external inputs.Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

References:

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

Inheritance

The contract for Mysterious Giant Dragon has the following inheritance structure.



Privileged Functions (onlyOwner)

Please Note if the contract is Renounced none of this functions can be executed.

Function Name	Parameters	Visibility
renounceOwnership		Public
transferOwnership	address newOwner	Public
changeWalletLimit	uint256 _amount	Public
changeSwapLimit	uint256 _maxBuyAmount, uint256 _maxSellAmount	Public
setCurrency	address _currency, address _router	Public
disableChangeTax		Public
disableWalletLimit		public
disableSwapLimit		External
launch		External
setkb	uint256 a	External
claimTokens		External

Function Name	Parameters	Visibility
setSwapAndLiquifyEnabled	bool _enabled	External
setNumTokensSellToAddToLiquidity	uint256 swapNumber	External
completeCustoms	uint256[] calldata customs)	External
setCurrency	address _currency, address _router	External
setFeeWhiteList		External
multi_bclist		Public
setSwapPairLisg		External
includeInReward		External
excludeFromReward		External
setSwapAndLiquifyEnabled	bool status	Public
setFundAddress		Public

Smart Contract Advance Checks

ID	Severity	Name	Result	Status
MGD-01	Low	Potential Sandwich Attacks.	Fail	Detected
MGD-02	Low	Function Visibility Optimization	Fail	Detected
MGD-03	High	Lack of Input Validation.	Fail	Detected
MGD-04	High	Centralized Risk In addLiquidity.	Fail	Detected
MGD-05	Low	Missing Event Emission.	Fail	Detected
MGD-06	Low	Conformance with Solidity Naming Conventions.	Pass	Not Detected
MGD-07	Low	State Variables could be Declared Constant.	Pass	Not Detected
MGD-08	Low	Dead Code Elimination.	Pass	Not Detected
MGD-09	High	Third Party Dependencies.	Pass	Not Detected
MGD-10	High	Initial Token Distribution.	Pass	Not Detected
MGD-11	High		Pass	Not Detected
MGD-12	High	Centralization Risks In The X Role	Pass	Not Detected
MGD-13	Informational	Extra Gas Cost For User..	Pass	Not Detected
MGD-14	Medium	Unnecessary Use Of SafeMath	Fail	Detected
MGD-15	Medium	Symbol Length Limitation due to Solidity Naming Standards.	Pass	Not Detected
MGD-16	Medium	Taxes can be up to 100%	Pass	Not Detected

ID	Severity	Name	Result	Status
MGD-17	Informational	Highly Permissive Role Access.,`	Pass	Not Detected
MGD-18	Informational	Stop Transactions by using Enable Trade.	Fail	Detected

MGD-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	🟡 Low	6-15-2023 InSpace.sol: L: 1464 C: 30	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.

Referrences:

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

MGD-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Gas Optimization	Low	6-15-2023 InSpace.sol: L: 125 C: 30	Detected

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
_swapPairList		internal
inSwapAndLiquify		internal
excludeFromReward		public
multi_bclist		public
setCurrency		public
setNumTokensSellToAddToLiquidity		public
setSwapAndLiquifyEnabled		public
claimTokens		public
setkb		public
launch		public
disableSwapLimit		public

Function Name	Parameters	Visibility
disableWalletLimit		public
disableChangeTax		public
setCurrency		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.

MGD-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	🟢 High	6-15-2023 InSpace.sol: L: 680 C: 11	Detected

Description

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

The given input is missing the check for the .

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. .

MGD-04 | Centralized Risk In addLiquidity.

Category	Severity	Location	Status
Coding Style	🟠 High	6-15-2023 InSpace.sol: L: 1518 C: 11	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 MGD-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

MGD-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	🟢 Low	6-15-2023 InSpace.sol: L: 327, 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.

MGD-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	🟡 Medium	6-15-2023 InSpace.sol: L: 109 C: 11	Detected

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations
 will automatically revert in case of integer overflow or underflow.
 library SafeMath {
 An implementation of SafeMath library is found.
 using SafeMath for uint256;
 SafeMath library is used for uint256 type in contract.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the
 Solidity programming language

Project Action

MGD-18 | Stop Transactions by using Enable Trade.

Category	Severity	Location	Status
Logical Issue	🔴 Informational	6-15-2023 InSpace.sol: L: 1265 C: 14	Detected

Description

Enable Trade is presend 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.
ℹ 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	2	0	0
🟢 Low	2	0	0
ℹ Informational	1	0	0
Total	7	0	0

Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/GuaGua568?s=09	Pass
Other		Fail
Website	https://guagua.pro/	Pass
Telegram	https://t.me/GuaGuaEN	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:

CFG NINJA

Assessment Results

Score Results

Review	Score
Overall Score	62/100
Auditor Score	61/100
Review by Section	**Score**
Manual Scan Score	14/53
SWC Scan Score	27 /37
Advance Check Score	21 /19

The Following Score System Has been Added to this page to help understand the value of the audit, the maximun score is 100, however to attain that value the project most 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:

- A few issues were identified, that need to be created.
- This is a factory contract from https://fatsale.finance.
- unfortunately, it does not pass the solidity standards.
- Please DYOR on the project.

Auditor Score =61
Audit Fail

Appendix

Finding Categories

Centralization / Privilege

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

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM 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 howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing 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 mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe 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 require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias 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 advocation 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.