Cyberscope

Audit Report

GAIA

July 2024

Networks	ETH, POLYGON, BSC, BASE, AVAX
Audited by	© cyberscope

Overview

GAIA is an Omnichain Fungible Token (OFT) deployed across five networks (ETH, POLYGON, BSC, BASE, AVAX). The tokenomics of GAIA involve allocating 40% of the total supply to each network's “staking” address, 10% to a “reward” address and reserving the remaining 50% in the GAIA contract. The latter is utilised for token sales, while the “staking” wallet is used to distribute rewards for participants in the staking process. Notably, the balance of the “reward” address is not actively used within the contract's operations. GAIA defines five price tiers for token sales in exchange for ETH, with incoming funds distributed among the community wallet, exchange wallet, liquidity wallet, dev wallet, marketing wallet and team wallet. The sale for each tier continues until its designated supply is exhausted. The latter sets an effective upper limit to the price of the token, until the reserved supply for every price tier is consumed. An effective measure against this would be to pause the token sale after a certain period.

GAIA also features a robust staking mechanism. Staking rewards are calculated linearly over time and are capped after one year. The staking process uses GAIA tokens, with rewards being distributed from the balance of the “staking” address. Users can unstake their funds at any time and claim rewards after a period of at least one day. Additionally, the claim function is automatically called for every existing user who re-stakes. This mechanism ensures a dynamic and user-friendly staking experience, promoting long-term engagement and token utility within the GAIA ecosystem.

Table of Contents

Risk Classification

The criticality of findings in Cyberscope’s smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorised into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	GAIA
Compiler Version	v0.8.22+commit.4fc1097e
Optimization	200 runs
Explorers	https://etherscan.io/address/0x7c61e1c20E57511831Ab02F03CA27aE4f558B80E https://polygonscan.com/address/0xb3C6063120d1eEeD32D4269C217e0f8C471E6270 https://bscscan.com/address/0xa927c2ca93ac9f5a0166e50784bcaa8e4c94323b https://basescan.org/address/0xA927C2ca93Ac9f5A0166E50784bCAa8e4c94323b https://snowtrace.io/address/0xA927C2ca93Ac9f5A0166E50784bCAa8e4c94323b
Addresses	0x7c61e1c20E57511831Ab02F03CA27aE4f558B80E 0xb3C6063120d1eEeD32D4269C217e0f8C471E6270 0xA927C2ca93Ac9f5A0166E50784bCAa8e4c94323b 0xA927C2ca93Ac9f5A0166E50784bCAa8e4c94323b 0xA927C2ca93Ac9f5A0166E50784bCAa8e4c94323b
Networks	ETH POLYGON BSC BASE AVAX
Symbol	GAIA

Decimals	18
Total Supply	20,000,000,000

Audit Updates

Initial Audit	19 Jul 2024

Source Files

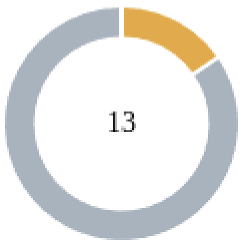
Filename	SHA256
contracts/GAIA.sol	f67f87bd2e789d2ab264a3557a188abbc52f03cda7876034cf1fa1ec5117eb82
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6
@openzeppelin/contracts/utils/Address.sol	b3710b1712637eb8c0df81912da3450da6ff67b0b3ed18146b033ed15b1aa3b9
@openzeppelin/contracts/utils/introspection/IERC165.sol	07ae1ac964ab74dedada999e2dfc642031a6495469cffc0bf715daa4f1e4f904
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/ERC20.sol	2d874da1c1478ed22a2d30dcf1a6ec0d09a13f897ca680d55fb49fbcc0e0c5b1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	471157c89111d7b9eab456b53ebe9042bc69504a64cb5cc980d38da9103379ae
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	912509e0e9bf74e0f8a8c92d031b5b26d2d35c6d4abf3f56251be1ea9ca946bf
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81
@layerzerolabs/lz-evm-protocol-v2/contracts/messagelib/libs/PacketV1Codec.sol	ed105f6e0e1ac7ef92b582e30d0d55d260cdcd6ca63057cd866f4c019c069b44

@layerzerolabs/lz-evm-protocol-v2/contracts/libs/AddressCast.sol	e48211bfba874d0ce5c3b1fc4e4aac67450ac9539a75e814807745181d04e7dd
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/ISendLib.sol	816acb88e015f130be9811b4e8efda8d829830ead2f009eb5a9444137af301b5
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/IMessagingContext.sol	b7f081828b8d0d206833953ebd73101bb28ac62541e7e8365fddc38eb440a92a
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/IMessagingComposer.sol	5e7e5a47a477dcf05c034d5135bf6516119af881ea78d586805eee0ad5ab6b12
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/IMessagingChannel.sol	9d0100f7cede83d5a4f6c9e03d6bfd24f861e9331b298ca4c73f99b161c718d5
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/IMessageLibManager.sol	ffc33e319ab5e9d1392d16f8716625bcfb1dc34b744551f87dad1c0eb3ad11f1
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/IMessageLib.sol	7a5733cf68663e1a924c5237adfdad5522a063e2214a81a7ca932242dc45481b
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/ILayerZeroReceiver.sol	24e4add7dd95a72dad23e26c4fadfc1b6ce63148e0c7f4d6ad5dfaeda79db3e7
@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/ILayerZeroEndpointV2.sol	a88fb702b371527c3791d7e5c168c1157d39021ed29b51e627e02973e6a740e0
@layerzerolabs/lz-evm-oapp-v2/contracts/precrime/OAppPreCrimeSimulator.sol	b0fcd1e2da22197e21ff13a350ac9d0a7f7e5e8bb1528c5d9860a3b820f555f9
@layerzerolabs/lz-evm-oapp-v2/contracts/precrime/libs/Packet.sol	b95b1e7f7470306f41da606d0984b9384d2134fc19ee814d7d9f98c613836745
@layerzerolabs/lz-evm-oapp-v2/contracts/precrime/interfaces/IPreCrime.sol	25a8f3b9257ed4f65fca276f7631cca3ead87c2f17c837e6eee90c8835433380
@layerzerolabs/lz-evm-oapp-v2/contracts/precrime/interfaces/IOAppPreCrimeSimulator.sol	2db19c9641e81e5366c268c40506db56ee5434b5c874cffe02fb1e18034a9285

@layerzerolabs/lz-evm-oapp-v2/contracts/oft/OFTCore.sol	6bab87b377303ccd9573f24f63b2b893948bfeeae26c026a393ae17d2baa883b
@layerzerolabs/lz-evm-oapp-v2/contracts/oft/OFT.sol	6f7e6b7d514cd2be4e6e5fa0827706289882a62c496786c9c08d5560e1c52415
@layerzerolabs/lz-evm-oapp-v2/contracts/oft/libs/OFTMsgCodec.sol	3e6dfff6595f8af1c1618a80e30efd16f117652e1fd73ffb1f6795dcc443d658
@layerzerolabs/lz-evm-oapp-v2/contracts/oft/libs/OFTComposeMsgCodec.sol	e73b5e5c53927a21e51dc7adaafd8d6fedf46e46b8ac084ca86f58c8e9697556
@layerzerolabs/lz-evm-oapp-v2/contracts/oft/interfaces/IOFT.sol	5c7365dbcb5f70acb3e81219d10c118368f786209a6fee589a97a87aea52c434
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/OAppSender.sol	276f5d551ae7e478fb0c56002acf5476e628748c1f5ba0d63ea051e93813a137
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/OAppReceiver.sol	8d1dd4a6a74c3d9704a078e6404dcf7aab688b750ea64e8331f873e253862bc0
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/OAppCore.sol	b0a3e30618f5d4c71732a2048d7c2928be399508d6ec622c9345915c92b3dd0a
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/OApp.sol	92b191c937374b6e76686245b7b98eb3bc0bfacff4d54b279b7ecf1321878e04
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/libs/OAppOptionsType3.sol	7414ab6a173705ac741772684d2bda00a0ed044c5e9faa3339a2dfe10ec867f3
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/interfaces/IOAppReceiver.sol	da4aff9322961c648e2221f7df203a44b375a397f2548b74679b3e9d0ae9addd
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/interfaces/IOAppOptionsType3.sol	c14d4f013045452583eae3baf14743e0628e24c146c3beea5a10b992b1171756
@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/interfaces/IOAppMsgInspector.sol	5bb1fd1038c648bff4e9c7b88f322c5cde41d355f03090ad1ec46f6712f09c4e

@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/interfaces/IOAppCore.sol	041dd35f0a4a22425614414c05cd5aae6ce7e4b5b86b0350045e57c0ed3cbf1b
@chainlink/contracts/src/v0.8/shared/interfaces/AggregatorV3Interface.sol	87fd1685e015c2001673b442c212f07847a64afa2f3b22bbbcb7f211ed293e88

Findings Breakdown



●	Critical	0
●	Medium	2
●	Minor / Informative	11

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	2	0	0	0
●	Minor / Informative	11	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	SAI	Staking Amount Inconsistency	Unresolved
🟠	TSI	Tokens Sufficiency Insurance	Unresolved
⚪	CO	Code Optimization	Unresolved
⚪	IEM	Inadequate Error Messages	Unresolved
⚪	MC	Missing Check	Unresolved
⚪	ODM	Oracle Decimal Mismatch	Unresolved
⚪	POSD	Potential Oracle Stale Data	Unresolved
⚪	TUU	Time Units Usage	Unresolved
⚪	UC	Unnecessary Calculations	Unresolved
⚪	UC	Unnecessary Calculations	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L17	Usage of Solidity Assembly	Unresolved
⚪	L19	Stable Compiler Version	Unresolved

SAI - Staking Amount Inconsistency

Criticality	Medium
Location	contracts/GAIA.sol#L120
Status	Unresolved

Description

The contract exhibits inconsistencies in handling staked amounts and expected rewards. These discrepancies can lead to potential loss of funds for some users and create overall inconsistencies.

In this case, the contract stores the staked amounts in the same address designated for reserving tokens for future staking rewards. Consequently, the subsequent check does not adequately ensure that the necessary reserves exist for the future rewards of a new staker.

```
require(balanceOf(staking) >= reserve + maxPossibleRewards, “...”);
```

As an example, we demonstrate the case that the sum of the expected rewards exceeds `balanceOf(staking)*365/1000`. In this case it is possible that some users are unable to withdraw the total of their staked balances.

User A calls: `stake(2_000_000*10**18)`

```
// Current State:
// --------------
 balanceOf(staking) = 8_000_000*10**18;
 reserve = 0;
// --------------
// New stake is accepted because:
// balanceOf(staking) >= reserve + maxPossibleRewards:
// 8_000_000*10**18 >= 0 + (2_000_000*10**18)*365/1000

// New State:
 balanceOf(staking) = 10_000_000*10**18;
 reserve = (2_000_000*10**18)*365/1000;
```

User B calls: `stake(7_000_000*10**18)`

```
// Current State:
// ---------------
balanceOf(staking) = 10_000_000*10**18;
reserve = (2_000_000*10**18)*365/1000;
// ---------------
// New stake is accepted because:
// balanceOf(staking) >= reserve + maxPossibleRewards:
// 10_000_000*10**18 >= (9_000_000*10**18)*365/1000
// New State:
balanceOf(staking) = 17_000_000*10**18;
reserve = (9_000_000*10**18)*365/1000;
```

After 1 year, User A calls `claim()` and User B also calls `claim()`:

```
// Current State:
balanceOf(staking) = 17_000_000*10**18
// New State:
balanceOf(staking) = (17_000_000-(9_000_000*365/1000))*10**18 =
                   = 13715000*10**18
```

However the sum of the staked amounts is equal to `15_000_000*10*18` which is more than `balanceOf(staking)` from the last state update.

This means that at least one of the users will not be able to claim their initial deposit and the expected staking rewards. The same scenario can be demonstrated with more users depositing smaller amounts.

Recommendation

The team should reassess their staking mechanism implementation, taking into account various edge case scenarios. A possible improvement would be to separate the staked amounts and the expected rewards into different addresses.

TSI - Tokens Sufficiency Insurance

Criticality	Medium
Location	contracts/GAIA.sol#L14
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
address public staking = 0xCaDc2B1bD1124BFb65A2fe05a948aa23303f353d;
```

```
require(balanceOf(staking) > 0
  ....
_transfer(staking, msg.sender, rewards);
```

The `staking` address plays a crucial role in the contract's staking mechanism. The balance of the `staking` address must be non-zero for staking to be activated. Staking rewards are also distributed to users from this address. The value of `staking` is defined in the constructor as `0xCaDc2B1bD1124BFb65A2fe05a948aa23303f353d`. At the time of the audit, this specified address is not a deployed contract with predefined logic. It is possible that it is an EOA wallet controlled by the team.

Recommendation

It is recommended to consider implementing a more decentralised and automated approach for handling the contract tokens. One approach would be to deploy a smart contract at the specified address, responsible for distributing the rewards.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L63
Status	Unresolved

Description

There are code segments that could be optimised. A segment may be optimised so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations. Specifically, the `getChainId()` method is called before every request to the Chainlink oracle to obtain the network's ID. However, since the network ID is constant and determined at deployment, this approach unnecessarily increases gas costs for users and adds complexity to the contract.

```
function getChainlinkDataFeedLatestAnswer() public view returns (int) {
uint256 chainId = getChainId();
address priceFeedAddress = chainlinkPriceFeedAddresses[chainId];
require(priceFeedAddress != address(0), "Price feed not available for
this chain");
AggregatorV3Interface dataFeed =
AggregatorV3Interface(priceFeedAddress);
(, int answer, , , ) = dataFeed.latestRoundData();
return answer;
}
```

Recommendation

The team is advised to review and optimise this code segment to enhance runtime performance. Specifically, the `getChainId()` method and the subsequent definition of the `priceFeedAddress` for the Chainlink Oracle could be called once from the constructor.

MC - Missing Check

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L159
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitised and checked that they form the proper shape. These variables may produce vulnerability issues.

The contract allows the `claim()` function to be called externally without verifying the balance of `staking` or whether staking is active.

```
function claim() public {
require(staked[msg.sender] > 0, "Aucun jeton n'est stake");
uint256 secondsStaked = block.timestamp - stakedFromTS[msg.sender];
...
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

IEM - Inadequate Error Messages

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L122,L123,127,L144,L145,L159,L187
Status	Unresolved

Description

The contract contains error messages that do not suitably describe the problems encountered. Specifically, the error messages lack clarity and specificity, making it difficult to identify and fix the issues. As a result, the users will not be able to find the root cause of the error. In this case, there exist code segments that return error messages written in a language other than English.

```
require(balanceOf(msg.sender) >= amount,
"Le solde est inferieur au montant");
```

Recommendation

The team is suggested to provide a descriptive message of the errors. Using English to display error messages enhances user experience, readability, and contract comprehension from the users.

ODM - Oracle Decimal Mismatch

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L85,L86
Status	Unresolved

Description

The contract relies on data retrieved from an external Oracle to make critical calculations. However, the contract does not include a verification step to align the decimal precision of the retrieved data with the precision expected by the contract's internal calculations. This potential mismatch in decimal precision can introduce substantial errors in calculations involving decimal values.

```
uint256 ethPrice = uint256(ethPriceUSD);
uint256 costInETH = (tierPriceUSD * amount * 10 ** 8) / ethPrice;
```

Specifically, the contract assumes the price provided by the Chainlink Oracle is retrieved with an 8-decimal precision. However, the contract does not implement a mechanism to verify the precision of the provided data.

Recommendation

The team is advised to retrieve the decimals precision from the Oracle API in order to proceed with the appropriate adjustments to the internal decimals representation. For more information visit Chainlink's Documentation https://docs.chain.link/data-feeds/api-reference

POSD - Potential Oracle Stale Data

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L71
Status	Unresolved

Description

The contract relies on retrieving price data from an oracle. However, it lacks proper checks to ensure the data is not stale. The absence of these checks can result in outdated price data being trusted, potentially leading to significant financial inaccuracies.

```
function getChainlinkDataFeedLatestAnswer() public view returns (int) {
 uint256 chainId = getChainId();
 address priceFeedAddress = chainlinkPriceFeedAddresses[chainId];
 require(priceFeedAddress != address(0),
 "Price feed not available for this chain");
 AggregatorV3Interface dataFeed =
AggregatorV3Interface(priceFeedAddress);
 (, int answer, , , ) = dataFeed.latestRoundData();
 return answer;
}
```

Recommendation

To mitigate the risk of using stale data, it is recommended to implement checks on the round and period values returned by the oracle's data retrieval function. The value indicating the most recent round or version of the data should confirm that the data is current. Additionally, the time at which the data was last updated should be checked against the current interval to ensure the data is fresh. For example, consider defining a threshold value, where if the difference between the current period and the data's last update period exceeds this threshold, the data should be considered stale and discarded, raising an appropriate error.

For contracts deployed on Layer-2 solutions, an additional check should be added to verify the sequencer's uptime. This involves integrating a boolean check to confirm the sequencer is operational before utilising oracle data. This ensures that during sequencer downtimes, any transactions relying on oracle data are reverted, preventing the use of outdated and potentially harmful data.

By incorporating these checks, the smart contract can ensure the reliability and accuracy of the price data it uses, safeguarding against potential financial discrepancies and enhancing overall security.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L124,L147,L162,L164,L165
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 daysStaked = secondsStaked / 86400;
```

Recommendation

It is good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

UC - Unnecessary Calculations

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L99
Status	Unresolved

Description

The contract includes instances of redundant calculations, such as the definition of a local variable with the same value as an existing one or performing trivial operations. These unnecessary calculations result in redundant code, reduced readability, and increased gas consumption, impacting the contract's performance. For example, declaring a state variable that merely mirrors another variable without serving an essential role in the contract's execution is inefficient and unnecessary.

In this case, the calculation of `totalCostShare` is a trivial operation,

```
uint256 totalCostShare = (totalCost * 100) / 100;
```

as `totalCostShare` will always be equal to `totalCost`. Once declared, the variable is only used in the local scope of the function and does not serve any other role.

Recommendation

The team is advised to consider this segment and rewrite it to ensure the state variable definition is meaningful for the contract's runtime.

UC - Unnecessary Calculations

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L95
Status	Unresolved

Description

The contract includes instances of redundant calculations, such as the definition of a local variable with the same value as an existing one or performing trivial operations. These unnecessary calculations result in redundant code, reduced readability, and increased gas consumption, impacting the contract's performance. For example, declaring a state variable that merely mirrors another variable without serving an essential role in the contract's execution is inefficient and unnecessary.

In this case, the calculation of `totalCost` is unnecessary, since its value can be derived from existing variables.

```
uint256 amount = (msg.value * 10 ** 18) / costInETH;
...
uint256 totalCost = (amount * costInETH) / 10 ** 18;
```

Specifically, substituting the equation for `amount` into the calculation of `totalCost` yields that:

```
uint256 totalCost = msg.value
```

The calculation of `totalCost` is therefore redundant.

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by utilising the existing variable declaration.

Refactoring the code reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L13,14,15,16,17,18,19,20,21,22,23
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address public reward = 0x8789ceC3967A64D78b28c4f71B16BdD01D48b6c9
address public staking = 0xCaDc2B1bD1124BFb65A2fe05a948aa23303f353d
address public communityWallet =
0xA87c43167e08C5Bd6Ae089153371b5cf72706d10
address public exchangesWallet =
0x3e3Cc7e2956674e334f695E34B2DD37B8449B16e
address public liquidityWallet =
0x2C7A0dd1aba8f972C6E87451c33D4629749d5966
address public devWallet = 0x00e28Dc9537316e0C279bf2a2E272a3e9A6752Dc
address public marketingWallet =
0xFb557DC3a559bcebAF38Fe3d77EEA2Dd518fecB1
address public teamWallet = 0x97a563bbcF6aAFA207fDA301E8d399D22aCeC03E
address public supportWallet =
0xA47924C6b3F5841F43Ff6e487247485505Fa3f1e
address public constructionWallet =
0xE3d784c80fBa7d773ec8D56c509c59B6E408a3Fa
address public tresorieWallet =

0x21cEb4150d5605C73a1Ef1e9801E87432c492fdB
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L65
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            chainId := chainid()
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/GAIA.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.22;
```

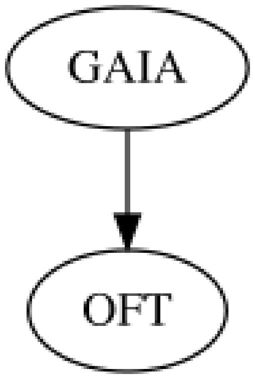
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

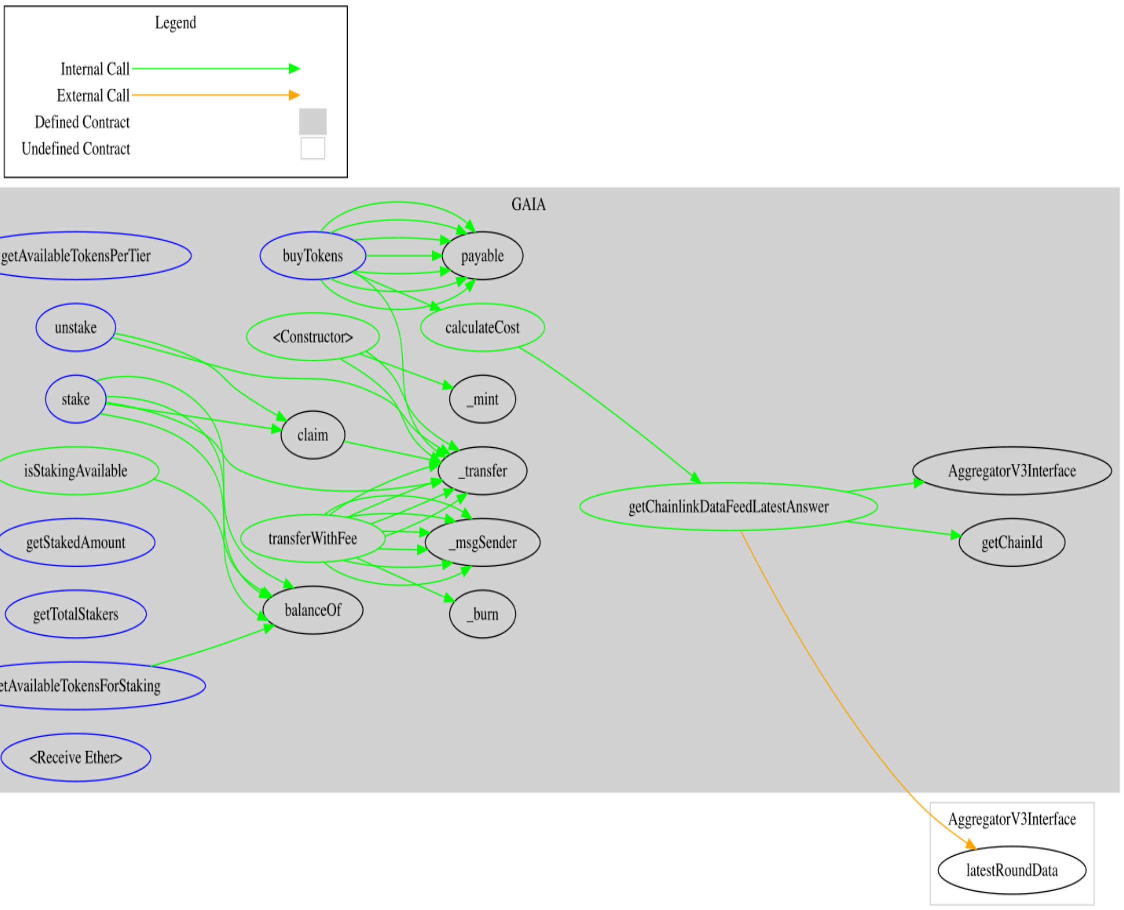
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
GAIA	Implementation	OFT		
		Public	✓	OFT Ownable
	getChainId	Public		-
	getChainlinkDataFeedLatestAnswer	Public		-
	calculateCost	Public		-
	buyTokens	External	Payable	-
	getAvailableTokensPerTier	External		-
	stake	External	✓	-
	unstake	External	✓	-
	claim	Public	✓	-
	isStakingAvailable	Public		-
	getStakedAmount	External		-
	transferWithFee	Public	✓	-
	getTotalStakers	External		-
	getAvailableTokensForStaking	External		-
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
GAIA
getAvailableTokensPerTier
buyTokens
payable
unstake
<Constructor>
calculateCost
stake
_mint
claim
isStakingAvailable
_transfer
AggregatorV3Interface
getChainlinkDataFeedLatestAnswer
getStakedAmount
transferWithFee
_msgSender
getChainId
getTotalStakers
balanceOf
_burn
getAvailableTokensForStaking
<Receive Ether>
AggregatorV3Interface
latestRoundData

Summary

GAIA is an Omnichain Fungible Token (OFT) deployed on five networks (ETH, POLYGON, BSC, BASE, AVAX). The tokenomics allocate 40% of the total supply to each network's “staking” address, 10% to a “reward” address, and retain 50% in the GAIA contract for token sales. The sales follow five price tiers and distribute incoming funds across several wallets, including community, liquidity, development, marketing, and team. The staking mechanism offers linear rewards capped after one year, with rewards distributed from the “staking” address. Users can stake, unstake, and claim rewards flexibly, enhancing engagement and utility within the GAIA ecosystem.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io