Autonolas Tokenomics

Smart Contract Audit Report

AUDIT SUMMARY

Autonolas Tokenomics builds upon the existing Autonolas platform to incentivize developers by distributing ETH donations and OLAS token top ups.

For this audit, we reviewed the project team's contracts folder at the post-audit branch on the team's private GitHub repository.

AUDIT FINDINGS

All findings have been resolved, though some centralized aspects are present.
Date: February 16th, 2023.
Updated: February 27th, 2023 with changes from the pre-audit branch to the post-audit branch.

Finding #1 - Depository - Informational (Resolved)

Description: *The _locked variable is never meaningfully used in the contract.*
Recommendation: *This variable can be removed to save on contract size and deployment cost.*
Resolution: *The team has removed this variable.*

CONTRACTS OVERVIEW

- *The contracts utilize ReentrancyGuard to protect against reentrancy attacks in applicable functions.*
- *As the contracts are implemented with Solidity v0.8, they are safe from any possible overflows/underflows.*
- *The contracts contain structs optimized for minimal storage slots and well-structured logic for gas savings.*

Depository Contract:

- *This contract allows users to deposit approved tokens for Bonds which can be redeemed for OLAS tokens.*
- *The owner may open a new "Bond Product" at any time.*
- *A Bond Product defines the valid purchasing token, the value of the token, the total supply of OLAS tokens that may be purchased, and how long the Bond Product is active.*
- *The purchasing token must be approved in the Treasury address.*
- *The team must exercise caution that purchasing tokens are 18 decimals.*
- *The Tokenomics address must approve the amount of purchasable OLAS tokens.*
- *The owner may close a Bond Product at any time.*
- *All remaining purchasable OLAS tokens will be "refunded" in the Tokenomics contract.*
- *Any address may purchase a Bond from an active Bond Product.*
- *The specified amount of tokens will be transferred to the Treasury address.*
- *The user's Bond will then be allocated an amount of OLAS tokens based on the value of the deposited tokens.*
- *The amount of OLAS tokens allocated is determined by the BondCalculator address.*
- *The Bond will be given a "maturity" time equal to the Bond Product's expiry time.*
- *Users may redeem their Bonds once the maturity time has passed.*
- *The users will be transferred OLAS tokens equal to the total payout of all redeemed Bonds.*

- *The owner may transfer ownership at any time.*
- *The owner may change the BondCalculator, Tokenomics and Treasury addresses at any time.*
- *The team must monitor approved tokens for large price swings as users will be able to purchase Bonds at a discount if a token's value drops below the Bond Product's set price.*

GenericBondCalculator Contract:

- *This contract is used to calculate the equivalent amount of OLAS tokens for another specified token.*
- *The OLAS payout is determined based on the total value of the specified tokens.*

Treasury Contract:

- *This contract is used to manage deposited tokens and donated ETH.*
- *Any ETH sent directly to the contract will be allocated to the team. Any ETH sent to the contract must be greater than the "minimum accepted ETH" amount.*
- *Any address may deposit a whitelisted token through the Depository contract.*
- *The tokens will be stored in the contract and the user will be minted an amount of OLAS tokens in return.*
- *The amount of OLAS tokens to be minted is determined in the Depositor contract.*
- *Any address may deposit ETH to a list of specified Services as a "service donation".*
- *The total amount of ETH donated must be greater than the minimum accepted ETH amount.*
- *The amount of ETH specified for each Service will be recorded and allocated in the Tokenomics contract.*
- *The Dispenser address may withdraw ETH deposited as a service donation to a specified address when the contract is not paused. This will additionally mint the address any OLAS token "top ups" they are due.*
- *The Tokenomics address may perform a "rebalance" when the contract is not paused.*
- *This will allocate a specified amount of ETH collected from service donations and reallocate it as raw ETH for the contract.*
- *The owner may "drain" any "slashed" funds from the Service Registry address at any time.*
- *The owner may withdraw any ETH sent to the contract not as a service donation at any time.*
- *The owner may withdraw any deposited tokens at any time.*
- *The owner may transfer ownership at any time.*
- *The owner may update the Tokenomics, Depository, and Dispenser addresses at any time.*
- *The owner may update the minimum accepted ETH value at any time.*
- *The owner may add and remove any token from the whitelist at any time.*
- *The owner may pause and unpause the contract at any time.*
- *The team must exercise caution when approving tokens to avoid using fee-on-transfer tokens.*

Dispenser Contract:

- *This contract allows users to claim rewards for their Agents and Components.*
- *Any address may claim rewards for specified Agents and Components that they own.*
- *The user's ETH rewards will be withdrawn from the Treasury and transferred to the user.*
- *Any additional OLAS top ups will be minted to the user.*
- *The owner may transfer ownership at any time.*
- *The owner may update the Tokenomics and Treasury addresses at any time.*

DonatorBlacklist Contract:

- *This contract is used to ban addresses from donating ETH to Services.*
- *The owner may add and remove any address from the blacklist at any time.*
- *The owner may transfer ownership at any time.*

Tokenomics Contract:

- *This contract is used to manage donations made to Services and the OLAS token's total supply used for Bonds and top ups.*
- *The OLAS total supply will increase by a set amount each year for the first 10 years after the token's "launch time".*
- *Each year after the first 10 years the token supply will increase by 2% each year.*
- *The new tokens will be allocated between top ups and Bond purchases.*

> - *The "max bond" amount of tokens reserved for Bond purchases each epoch is determined by the "max bond fraction".*
> - *The remaining tokens will be split between top ups for Agent owners and Component owners.*
> - *The percentage of tokens allocated to each is determined by the "top up unit fraction" for the Agents and Components respectively.*
> - *Any address may perform a "checkpoint" if it has been at least the "current epoch length" and less than a year since the last checkpoint.*
> - *Performing a checkpoint will apply the inflation that has occurred from the last checkpoint and begin the next epoch.*
> - *The max bond will be recalculated using the new max bond fraction if it was updated during the current epoch.*
> - *The amount of tokens allocated to Bond purchases will be increased by the max bond amount with any additional inflation that has occurred after the current epoch length.*
> - *The next epoch's minimum length and the "veOLASThreshold" values will be changed if they were updated during the current epoch.*
> - *The current "inverse discount factor" will be updated if any donations to Services were received during the current epoch.*
> - *The inverse discount factor is determined based on the "productivity" of the platform as determined by the "code per dev", "devs per capital", and the number of new Agent or Component owners who have received rewards.*
> - *The inverse discount factor cannot exceed the current "epsilon" value.*
> - *As productivity increases users who purchase Bonds will receive more tokens.*
> - *The portion of the epoch's inflation not reserved for Bond purchases will be allocated to top ups within the Treasury contract.*
> - *The "reward treasury fraction" of ETH donations received during the epoch will be "rebalanced" for the Dev team.*
> - *The checkpoint will fail if the Treasury is unable to properly rebalance.*
> - *The inverse discount factor will reset back to 1 if no donations were received.*
> - *Any address may donate ETH to specified Services through the Treasury contract.*
> - *The ETH will be allocated evenly across Unit types within the Service; all Agents will receive the same share and all Components will receive the same share.*
> - *The ETH allocation is determined by the "reward unit fraction" for Agents and Components respectively.*
> - *The allocated ETH will be added to the Unit owner's pending rewards for the epoch.*
> - *Unit owners are eligible for top ups if they have staked a sufficient amount of OLAS tokens for veOLAS tokens.*
> - *Unit owners will earn top ups from the total inflation allocated to top ups equal to the fraction of total ETH donations they receive.*
> - *Rewards are claimable after they are "finalized" in a later epoch.*
> - *Unit owners may claim their rewards through the Dispenser contract.*
> - *The Depository address may reserve and refund OLAS tokens from the max bond amount at any time.*
> - *The owner may transfer ownership at any time.*
> - *The owner may update the Donator Blacklist, Treasury, Depository, and Dispenser addresses at any time.*
> - *The owner may update the Agent Registry, Component Registry, and Service Registry addresses at any time.*
> - *The owner may update the devs per capital, code per dev, and epsilon values at any time.*
> - *The owner may update the minimum epoch length and veOLAS threshold values at any time.*
> - *The owner may update the reward Component fraction and reward Agent fraction values at any time.*
> - *The owner may update the max bond fraction, top up component fraction, and top up agent fraction values at any time.*

AUDIT RESULTS

Vulnerability Category	Notes	Result
Arbitrary Jump/Storage Write	N/A	PASS
Centralization of Control	• The Treasury owner may add and remove approved tokens. • The DonatorBlacklist owner may add and remove any address from the blacklist. • The Tokenomics owner may update the reward distribution fractions. • The Tokenomics owner may update the top ups and Bond distribution fractions. • The Tokenomics owner may update the devs per capital, code per dev, and epsilon values.	WARNING
Compiler Issues	N/A	PASS
Delegate Call to Untrusted Contract	N/A	PASS
Dependence on Predictable Variables	N/A	PASS
Ether/Token Theft	N/A	PASS
Flash Loans	N/A	PASS
Front Running	N/A	PASS
Improper Events	N/A	PASS
Improper Authorization Scheme	N/A	PASS
Integer Over/Underflow	N/A	PASS
Logical Issues	N/A	PASS
Oracle Issues	N/A	PASS
Outdated Compiler Version	N/A	PASS
Race Conditions	N/A	PASS
Reentrancy	N/A	PASS
Signature Issues	N/A	PASS
Sybil Attack	N/A	PASS
Unbounded Loops	N/A	PASS
Unused Code	N/A	PASS
Overall Contract Safety		PASS

CONTRACT SOURCE SUMMARY AND VISUALIZATIONS

Name	Address/Source Code	Visualized (Hover-Zoom Recommended)
Depository	GitHub (Not yet deployed on mainnet)	Inheritance Chart. Function Graph.
GenericBondCalculator	GitHub (Not yet deployed on mainnet)	Inheritance Chart. Function Graph.
Treasury	GitHub (Not yet deployed on mainnet)	Inheritance Chart. Function Graph.
Dispenser	GitHub (Not yet deployed on mainnet)	Inheritance Chart. Function Graph.
DonatorBlacklist	GitHub (Not yet deployed on mainnet)	Inheritance Chart. Function Graph.
Tokenomics	GitHub (Not yet deployed on mainnet)	Inheritance Chart. Function Graph.

WHAT IS A SOLIDITY AUDIT?

Typically, a smart contract audit is a comprehensive review process designed to discover logical errors, security vulnerabilities, and optimization opportunities within code. A *Solidity Audit* takes this a step further by verifying economic logic to ensure the stability of smart contracts and highlighting privileged functionality to create a report that is easy to understand for developers and community members alike.

HOW DO I INTERPRET THE FINDINGS?

Each of our Findings will be labeled with a Severity level. We always recommend the team resolve High, Medium, and Low severity findings prior to deploying the code to the mainnet. Here is a breakdown on what each Severity level means for the project:

- **High** severity indicates that the issue puts a large number of users' funds at risk and has a high probability of exploitation, or the smart contract contains serious logical issues which can prevent the code from operating as intended.
- **Medium** severity issues are those which place at least some users' funds at risk and has a medium to high probability of exploitation.
- **Low** severity issues have a relatively minor risk association; these issues have a low probability of occurring or may have a minimal impact.
- **Informational** issues pose no immediate risk, but inform the project team of opportunities for gas optimizations and following smart contract security best practices.