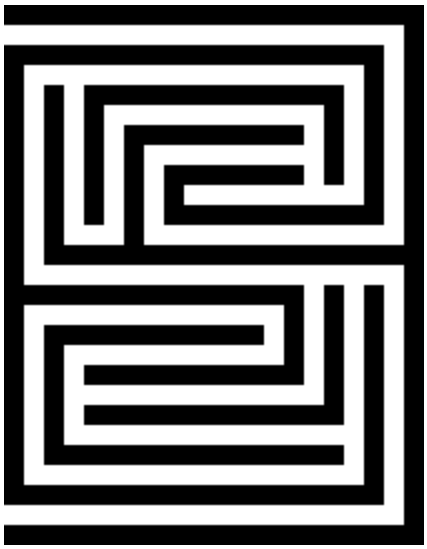

Aurox LLC. Public Report

PROJECT: Aurox LLC.

February 2021

Prepared For:

Giorgi Khazaradze | Aurox LLC.

giorgi@getaurox.com

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Aurox LLC. Protocol Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in February of 2021 to perform a comprehensive security review of the Aurox LLC. smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of three days by a member of the Bramah Systems, LLC. executive staff.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: February 10th, 2021

Report Delivery: February 15th, 2021

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Aurox LLC. protocol, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Contract Specification

Contract specification was provided in the form of code comments and functional unit tests, along with a verbose specification document which provided justification for infrastructure decisions and structural fundamentals.

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the Aurox LLC. Protocol. During the course of our engagement, Bramah Systems found few instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT, as our report details.

The team otherwise used **thoroughly** reviewed and vetted components and provided details as to the token structure, economics, and intent, which helped Bramah highlight any potential concerns with their approach.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the Aurox LLC. Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.
The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the Aurox LLC. protocol or any other relevant product, service or asset of Aurox LLC. or otherwise. This report is not and should not be relied upon by Aurox LLC. or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.
To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the Aurox LLC. Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

TODO remain in code

The following item exists within the StakingMaster.sol code, indicating that functionality remains to be developed.

// TODO import the interfaces instead to reduce the amount of code

Resolution: This line has been removed from the source code.

Provide explanation for magic number in code comment

The following magic number exists within **StakingMaster.sol**

uint256 private secondsPerMonth = 2628334;

As months differ in length and therefore do not have a set number of seconds, the variable appears to be the average seconds per month -- however, it does not appear to be correct. The average month is 30.42 days. A day is 24 hours, so the average month is 730.08 hours (30.42 days * 24 hours). 730.08 hours is equal to 43,804.8 minutes (730.08 hours * 60 minutes), or 2,628,288 seconds (43 , 804.8 minutes * 60 seconds).

Resolution: The Aurox team provided the following update (rendered below), which Bramah believes sufficiently addresses the finding.

> *The above calculation is more correct than the initial value, but it is still slightly off. According to the following link: https://www.rapidtables.com/calc/time/seconds-in-year.html the*
>
> *number of seconds in a Gregorian calendar year is 31556952.*
>
> *This equates to roughly 365.2425 days, the reason for this extra .2425 is to account for leap years. Based on this calculation the correct amount of seconds per month is 2,629,746.*
>
> *The secondsPerMonth value in the contract has been updated to reflect that.*

Unused local variable

Unused local variable **currentEpoch** exists within the **claimRewards** function of **Provider.sol.** As the unused variable does appear to serve a necessary function in a later function call, this variable should be properly initialized and utilized.

Resolution: This variable has been removed from the source code.

Function state can be pure

The function **_returnEpochAmountIncludingShare** can have a “pure” state rather than a “view” state as the function performs no actions which impact state.

Resolution: Function state has been appropriately changed.

Variable shadowing in function returnAllClaimableRewardAmounts

The function **returnAllClaimableRewardAmounts** has two variables, **rewardTotal** and **lastLiquidityAddedEpochReference** that shadow inherited state variables. These variables should be renamed where possible to prevent improper usage.

Resolution: While Bramah still suggests variable renaming, the Aurox team has provided the following justification which Bramah feels addresses relevant concerns:

As the shadowed variables exist inside a mapping -> structs, this shouldn’t require changing as their accessing must always be through their mapping and the local variables will then never be shadowed.

Commented out code in function claimRewards

The function **claimRewards** has the following line of commented-out code, which should be removed as it serves no purpose.

// require(allClaimableAmounts > 0, "No rewards to claim for the user");

Resolution: This line has been removed.

Specific Recommendations

Unique to the Aurox LLC. Protocol

Highly permissive owner account and centralization of power

The deploying account possesses a number of highly actions (namely, initiating protocol defaults and modifying payout parameters). This deploying account should (where possible) minimize usage of the associated key (e.g. performing transactions, using as a regular user account) and perform other operational security best practices. Potentially, this could involve transferring ownership to a MultiSignature governance.

Resolution: The Aurox team provides the following, which Bramah feels sufficiently addresses the concern raised:

> *To resolve this it is optimal to create a Multi-Signature wallet and once all contracts are deployed to transfer the owner of those contracts to the Multi-Signature wallet.*
>
> *The Aurox Token, Staking Master and Provider are all Ownable inherited contracts, this is an accepted standard that allows the ownership of those contracts to be transferred at any time. Once deployment is complete it is recommended to transfer the ownership to the Multi-Signature wallet. This is achieved through a function call on each of the contracts.*

Contract relies upon external contract not controlled by team

The contract relies upon an ERC-1167 cloning factory, referenced in the code as an external call-out to a contract (**VestingFactoy.sol**, Lines 36-51). This address references a deployed version of the clone-factory contracts, which notably currently have a build failure and have not been updated in 2 years, and most importantly - are not controlled by the team.

Resolution: The Aurox team provided the following:

> *The suggestion assumes that the Vesting Factory contract utilises a Third-Party deployment of the clone-factory contracts, this is incorrect as the deployment process deploys an instance of this factory and that is used within the contracts. Despite being 2 years old, the contract has been tested for its reliability and it behaves as expected.*

While the team does **deploy** their own version of the contract, it does so through the **createClone** function (referenced below). This function **does** reference the external address as referenced by Bramah and would be influenced by its logic,

```
function createClone(address target) internal returns (address result) {
        bytes20 targetBytes = bytes20(target);
        assembly {
            let clone := mload(0x40)
            mstore(
                clone,

0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000
            )
            mstore(add(clone, 0x14), targetBytes)
            mstore(
                add(clone, 0x28),

0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000
            )
            result := create(0, clone, 0x37)
        }
    }
```

After further discussion with the team, Bramah received the following notice: “Bramah is correct about the referencing of contracts, these referenced contracts are known as Minimal

Proxy contracts. These contracts were developed as a part of the EIP 1167 implementation and allow vastly reduced gas costs on deployment of factory contracts.

They are widely used within various clone-factory implementations and continue to be referenced since their deployment in 2018. Bramah is correct in them not being controlled by the team, but they are known to have a fixed and unchangeable byte-code implementation that no entity has access to.

The implementation of the contracts and rationale can be found here: https://eips.ethereum.org/EIPS/eip-1167”

Final discussion with the team resulted in a confirmation between the two parties that while risk exists from this reliance, given the longevity of the present place contracts, this risk does not rise to a high enough level to result in an overhaul and redeployment.

Design principles rely upon a “closed” system

By design, many principles within the protocol rely upon having a closed system design, wherein various functionality exists within a “wrapper” in lieu of the native functionality supported by the ERC20 token.

While this is an intentional design choice and used to facilitate proper execution of the contracts, users should be aware that these functions may perform differently than their ERC20 counterparts (e.g. the performance of interactions within token vesting). It is suggested that due to this, **ReentrancyGuard** or a similar framework be used.

Resolution: **ReentrancyGuard** has been added to all applicable functions.

Focus on “seconds” should be avoided

The contracts make extensive use of seconds (the atomic unit of time) and **block.timestamp**. Bramah suggests refocusing to larger time increments, as Solidity has multiple known caveats with time sensitive actions (largely relating to the passage of time and how miners may report it).

Resolution: Aurox has provided additional code comments around the usage.

Test coverage in most areas is poor

The test coverage in certain areas of the protocol (namely the token) is quite poor and should

be made further exhaustive to better reflect the teams intent for each function.

75% Statements `336/448`

58.08% Branches `115/198`

61.46% Functions `59/96`

74.78% Lines `338/452`

File	Statements		Branches		Functions		Lines	
Provider/	90.96%	161/177	85.53%	65/76	91.3%	21/23	90.91%	160/176
StakingMaster/	73.64%	95/129	47.14%	33/70	70.59%	12/17	73.48%	97/132
Token/	44.23%	23/52	20%	4/20	33.33%	6/18	44.23%	23/52
Uniswap/	100%	0/0	100%	0/0	100%	0/0	100%	0/0
Vesting/	63.27%	31/49	40%	8/20	52.63%	10/19	62.75%	32/51
lib/	63.41%	26/41	41.67%	5/12	52.63%	10/19	63.41%	26/41

Resolution: Aurox provided further configuration details as to their test suite and informed us

that due to the number of tests, the test coverage runner is unable to run for the entire duration as required. Individually performing test coverage scans on each contract results in **93%+** test coverage, which is far more acceptable and in-line with industry standards.

Toolset Warnings

Unique to the Aurox LLC. Protocol

Overview

In addition to our manual review, our process involves utilizing static analysis and formal methods in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No warnings were present at time of compilation.

Test Coverage

The contract repository possesses extensive unit test coverage throughout. This testing provides a variety of unit tests which encompass the various operational stages of the contract.

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril

- Oyente
- Slither

In each case, the team had either mitigated relevant concerns raised by each of these tools or provided adequate justification for the risk (such as adhering to the ERC-20 standard), or a concern stemming from the discovered risk was elevated to a larger issue and is referenced above.

Directory Structure

At time of review, the directory structure of the Aurox LLC. smart contracts repository appeared as it does below. Our review, at request of Aurox LLC., covers the Solidity code (*.sol) as of commit-hash **e5dc484** of the Aurox LLC. repository.

```
.
├── Migrations.sol
├── Provider
│   ├── IProvider.sol
│   ├── Provider.sol
│   └── artifacts
│       ├── Provider.json
│       └── Provider_metadata.json
├── StakingMaster
│   ├── IStakingMaster.sol
│   ├── StakingMaster.sol
│   └── artifacts
│       ├── AuroxToken.json
│       ├── AuroxToken_metadata.json
│       ├── StakingMaster.json
│       └── StakingMaster_metadata.json
├── TestHelpers
│   ├── ERC20.sol
```

```
│   └── ERC20Mintable.sol
├── Token
│   ├── AuroxToken.sol
│   ├── IAuroxToken.sol
│   ├── TokenVesting.sol
│   └── artifacts
│       ├── AuroxToken.json
│       ├── AuroxToken_metadata.json
│       ├── TokenVesting.json
│       └── TokenVesting_metadata.json
├── Uniswap
│   ├── IUniswapV2Factory.sol
│   └── IUniswapV2Router02.sol
├── Vesting
│   ├── TokenVesting.sol
│   ├── VestingFactory.sol
│   └── artifacts
│       ├── VestingFactory.json
│       └── VestingFactory_metadata.json
└── artifacts
    ├── AuroxToken.json
    ├── AuroxToken_metadata.json
    ├── AuroxVesting.json
    ├── AuroxVesting_metadata.json
    ├── VestingVault.json
    └── VestingVault_metadata.json

11 directories, 32 files
```