GAU Token & Vesting Smart Contract Security Audit

MINDSTONE

This report was prepared to explore issues and vulnerabilities in the source code of the Gamer Arena Utilily Token & Vesting smart contract, as well as contract dependencies that are not part of an officially recognized library. A comprehensive rev using Static Analysis and Manual Review techniques.

The audit process pays particular attention to:

● Testing smart contracts against both common and uncommon attack vectors.

● Evaluate the code base to ensure compliance with current best practices and industry standards.

● Ensure that the contract logic meets the customer's specifications and intentions.

● Cross-comparison of contract structure and development against similar smart contracts produced by industry leaders.

● Comprehensive line-by-line manual review of the entire code base by industry experts.

Our security assessment resulted in findings ranging from critical to informative. We recommend addressing these findings to ensure a high level of security standards and industry practices. We offer some suggestions that might serve the project better from a security standpoint.

● Improve overall coding processes for better structure of source code

● Add enough unit tests to cover possible use cases

● Since the contract will be publicly verified, each function must be checked for readability. Provide more comments

● After the protocol is published, other communication such as Telegram, Twitter, Discord. Provide more transparency through your channels

CATEGORIES

Centralization / Exceptions

These findings point to logic or practices that have been constructed to depart from the principles of decentralization. Special access rights and contract ownership that can be used in the transfer of assets are included in this section.

Gas Optimization

The Gas Optimization findings do not affect the functionality of the code, but suggest different, more optimal EVM transaction codes that result in a reduction in the overall gas cost of a transaction.

Mathematical Operations

It is related to the incorrect use of mathematical formulas such as findings, overflows, incorrect operations, etc.

Logical Problems

The Logical Issue findings detail an error in the logic of the linked code, such as a false assumptions on how the EVM works.

Temporary Code

These findings refer to sections of code that behave unexpectedly in certain extreme situations that could lead to a security vulnerability.

Coding Style

Coding Style findings usually don't affect the generated bytecode, but instead comment on how to make the codebase more readable and ultimately more maintainable.

RESULTS

According to our inspections, we have determined **10** points that need revision

SEVERITY	#
CRITICAL	0
MAJOR	2
NOTIFICATION	8

CONTRACTS

ERC-20 Type GauTestToken Contract

1. Centralization Risk

[MITIGATED]

In the contract, there are privileged functions like pause and unpause than can only be called by the contract owner. This creates risks, where the theft of the private keys of the owner account can damage the reputation of the project. This risk should be carefully managed.

In general, centralized privileges in the protocol shold be avoided. Making the owner a 2⁄3 or 3⁄5 multi-signature account is recommended. Using a multi-signature wallet with hardware wallets from different manufacturers is suggested. This approach is resistant to to vulnerabilities such as theft or loss.

Another suggestion is the implementation of time locks with reasonable delay. Owner privileged transactions cannot be made for 48 hours on changes in the account. It gives time to investors and technical team to intervene in the problem.

In the long run a DAO with voting/administration mechanism can be used. These approaches increase participation and transparency. Finally, the rights to ownership can be renounced.

2. Delegation requirement

[TAKEN INTO CONSIDERATION]

The contract uses the ERC20Votes extension, that is a more generic version of Compound’s voting and delegation mechanism. Compound’s voting mechanism has an upper limit of 296 - 1 whereas this version has an upper limit of 2224 - 1. The extension keeps a history of each user’s voting power called checkpoints. Users can transfer, delegate, their voting power either directly or by providing a signature to be used with the “delegatebySig” call. Current voting power and earlier voting powers can be queried. By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that it requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked.

The token source has been generated using the Open Zeppelin wizard and utilizes the ERC20Votes extension along with the implementation of the ERC20 Permit extension defined by EIP-2612. The extension allows approvals to be made via signatures. The permit method allowss to change the allowance of an ERC20 token by presenting a message signed by the account. The holder account doesn’t need to send a transaction and hold ETH at all.

3. Test Scenarios

[MITIGATED]

Writing test scenarios for the contract at both smart contract and dapp level and testing them first on the test network and then on the real network is extremely important. It will be important to build trust to show that these tests are also called in the real network and the results are produced correctly. It is recommended to write tests for the project. When the tests fail, it will help you to see exactly which scenarios do not meet your expectations and make sure that they are working.

4. ERC-20 Withdrawal

[TAKEN INTO CONSIDERATION]

The token contract does not have a method for withdrawing the ERC-20 token,in case it is sent to the contract address. Either adding a withdrawal method or preventing the sending of the tokens to the contract address is advised. For prevention the transfer function can be overridden, for withdrawal an owner controlled method should give access to the contract balance.

Vesting Contract

5. Unnecessary Safe Math Usage

[MITIGATED]

Solidity >= 0.8.0 provides errors for buffer overflow and underflow. There is no need to use SafeMath anymore. We recommend removing it.

6. Floating Pragma

[MITIGATED]

Contracts should be deployed with the same compiler version and flags that have been tested thoroughly. Locking the Pragma helps 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.

We recommend using a fixed version of the compiler (^ symbol should be removed from Pragma). Consider using the same compiler version for all contracts.

7. Public Function Can Be Declared External

[MITIGATED]

Public functions that are never called by the contract should be declared external to save Gas: pause, unpause, createVestingSchedule, revoke, withdraw, computeReleasableAmount, getWithdrawableAmount, computeNextVestingScheduleIdForHolder, getLastVestingScheduleForHolder. We recommend using the external attribute for functions never called from the contract.

8. Zero address is allowed

[MITIGATED]

The new address for the service signer does not check if it is a zero address, which could be sent as a default value.

We recommend adding a check for zero address for _beneficiary

9. Redundant payable address cast

[MITIGATED]

Release function casts beneficiary address to payable, which is redundant, as contract transfer ERC20 token, not the network native token.

We recommend removing the payable cast before the transfer.

10. Missing vesting validation

[MITIGATED]

createVestingSchedule function does not validate if the cliff period is less than the vesting duration. If the cliff is bigger than the duration - nothing would be released to the beneficiary before the cliff is ended.

We recommend validating cliff duration when creating a vesting schedule.

DISCLAIMER

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.