LeewayHertz - Software Development Company

GENI AI

Security Assessment

28 March 2023

Score: 91/100 (Safe)

For :

Geni AI

By :

sales@leewayhertz.com

OVERVIEW

Project Summary

Project Name	Geni AI
Description	A protocol with AutoStaking technology - generating tokens for its holders just by holding Geni tokens.
Platform	Arbitrum; Solidity
Contract	0x02ed10579e7acfda2923fd3084c145834f9e54ab

Audit Summary

Delivery Date	March, 28, 2023
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	March 26, 2023 - March 28, 2023

Vulnerability Summary

Total Issues	9
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	10

EXECUTIVE SUMMARY

Our team conducted a comprehensive analysis of the codebase for the Geni AI Protocol project. We found that the development team has implemented the latest security standards, including important security principles such as preventing funds from remaining at rest, which greatly enhances the security of the contracts. Despite our efforts to identify potential attack vectors, we were unable to find any vulnerabilities that could be exploited. We also provided optimization recommendations to the team, but after a collaborative discussion, it was concluded that the proposed changes were not significant optimizations and would require changes across the entire codebase, hence were not implemented.

FINDINGS

ID	Title	Type	Severity
SPA-01	Unused Function Call Results	Implementation	Informational
SPA-02	Potentially Redundant `bool` Variable Return	Implementation	Informational
SPA-03	Usage of `tx.origin`	Implementation	Minor
SPA-04	Variable State Mutability	Optimization	Informational
SPA-05	Variable Visibility Specifier	Syntax	Informational
SPA-06	Variable State Mutability	Optimization	Informational
SPA-07	BEP-20 `approve` Race Condition	Implementation	Informational
SPA-08	Contract Bytecode Optimization	Optimization	Informational

SPA-01: Unused Function Call Results

Type	Severity	Location
Implementation	Informational	Router L237, L238, L261, L485, L492, L573, L577, L590

Description:

The linked code segments are performing BEP-20 compliant `transfer` operations on arbitrary tokens. The standard denotes that a return variable should be set that dictates whether the `transfer` operation succeeded.

Recommendation:

Certain tokens that are otherwise BEP-20 compliant fail to satisfy this requirement and may fail if a strict check is imposed and as such, we advise to utilize a fork of the OpenZeppelin `SafeERC20` library that is minimally adapted to work with BEP-20 tokens.

The library essentially renders the return variable optional, however should it exist it should evaluate to `true`.

SPA-02: Potentially Redundant `bool` Variable Return

Type	Severity	Location
Implementation	Informational	Router L197-L200

Description:

The linked code segment conducts a `transfer` operation, however it utilizes a non-standard `transferTo` function name and returns a `bool` variable that is always `true`.

Recommendation:

As the return variable remains unused within the contract, we advise that it is omitted unless the contract is meant to implement a standard that is undocumented within the code or the project's `README.md` file.

SPA-03: Usage of `tx.origin`

Type	Severity	Location
Implementation	Minor	Pool L198

Description:

The linked code segment conducts a token transfer between the original transaction invocator and the input `recipient` variable.

Recommendation:

While this design choice was made so that the removal of liquidity is possible instantenously, this is an invalid design paradigm. As `tx.origin` is utilized, any contract on the Arbitrum chain is capable of redirecting its calls to the `Pool` contract and consequently draining the funds of the original invocator.

We reached out to Arbitrum and were informed that smart contracts on the Arbitrum chain do not need an audit before being deployed on the mainnet, rendering this attack vector as `Medium` in severity as a tangible attack vector exists.

Smart contracts on the Binance chain have the luxury of being fully audited before being deployed on the main-net, meaning this attack vector is `Minor`, however a smart contract can purposefully avoid "detection" from an audit by allowing an arbitrary function call execution i.e. governance systems. As such, this is a tangible attack vector.

We advise that either `msg.sender` is utilized and the workflows on `Router.sol` are revamped, or that a conditional is imposed that utilizes `tx.origin` only if the invoker of the function is the `Router` contract and otherwise utilizes `msg.sender`.

SPA-04: Variable State Mutability

Type	Severity	Location
Optimization	Informational	Pool L114, L115, L122, L123

Description:

The `BASE`, `TOKEN`, `decimals` and `genesis` variables are only assigned to once during the contract's creation.

Recommendation:

As their naming convention implies, they are meant to remain unchanged throughout the lifetime of the contract apart from their original definition during the contract's creation.

As such, we advise that the `immutable` mutability specifier is set on both variables to firstly ensure that their naming convention is properly reflected in code and lastly to greatly optimize the gas cost involved in utilizing them as they are hot-swapped directly in the code as `memory` reads rather than the `storage` reads they currently do.

We advise that the same optimization is applied to the `genesis` variable even though it is meant to be in lower-case to conform to the interface specification.

Finally, the `decimals` variable can directly be declared as a `constant` as it is statically assigned to the value literal of `18`.

SPA-05: Variable Visibility Specifier

Type	Severity	Location
Syntax	Informational	Pool L90

Description:

The linked variables contain no explicit visibility specifiers set.

Recommendation:

We advise that an explicit visibility specifier is set for those variables to aid the readers of the codebase and streamline compiler upgrades as each compiler can set a different default visibility specifier.

SPA-06: Variable State Mutability

Type	Severity	Location
Optimization	Informational	Pool L373, L374, L375

Description:

The `BASE`, `Eth` and `DEPLOYER` variables are only assigned to once during the contract's creation.

Recommendation:

As their naming convention implies, they are meant to remain unchanged throughout the lifetime of the contract apart from their original definition during the contract's creation.

As such, we advise that the `immutable` mutability specifier is set on both variables to firstly ensure that their naming convention is properly reflected in code and lastly to greatly optimize the gas cost involved in utilizing them as they are hot-swapped directly in the code as `memory` reads rather than the `storage` reads they currently do.

SPA-07: BEP-20 `approve` Race Condition

Type	Severity	Location
Implementation	Informational	Pool L145-L153

Description:

The `approve` function as built by both the ERC-20 and BEP-20 standard is inherently flawed in that it provides a race-condition attack vector as defined in the Smart Contract Weakness Classification registry under no. 114.

Recommendation:

There are multiple approaches to alleviating this attack vector that are entirely up to the developers. Potential approaches include `require`ing that the previously set approval value was `0` or that new functions that increment and decrement the allowance respectively are utilized such as `increaseAllowance` and `decreaseAllowance`. We list this finding as `Informational` despite its severity as it is a widely known issue in the Ethereum community and oft unconsidered.

SPA-08: Contract Bytecode Optimization

Type	Severity	Location
Optimization	Informational	Pool L265-L282, L283-L302

Description:

The linked function pairs `_getAddedBaseAmount` & `_getAddedTokenAmount` and `_swapBaseToToken` & `_swapTokenToBase` contain similar statements and can be optimized.

Recommendation:

We advise that a single function implementation is utilized instead that accepts an `iBEP20` argument denoting the contract to query the new balance from as well as a `uint256` argument that denotes the old balance which is either `_baseBalance` or `_tokenBalance` respectively.

Additionally, the `else` branch as well as explicit `return` statement can be omitted from the function as Solidity data types begin with their default zeroed out value and named return variables are automatically returned at the end of the function they are declared in.

The `_swapBaseToToken` and `_swapTokenToBase` functions can be merged into a single function and optimized identically.