Security Audit Report

08/03/2022

Genome Mining PR #49

RED4SEC

Content

Introduction

Altered State Machine is a platform for the creation and training of A.I. Agents, owned and traded using NFTs. ASM aims to be a new generation of metaverses and gaming, where AI agents compete with each other, interact, and support human actors.

Altered State Machine is a decentralized protocol for creating ownership of an AI agent via an NFT. ASM can be used to create Agents for games, financial applications, virtual assistants, online worlds, and many more.

As solicited by **Altered State Machine** and as part of the vulnerability review and management process, Red4Sec has been requested to perform a security code audit in order to evaluate the security of the **Genome Mining PR #49** project.

The report includes specifics retrieved from the audit for all the existing vulnerabilities of **Genome Mining PR #49**. The performed analysis shows that the smart contract does not contain any critical vulnerabilities.

Disclaimer

This document only represents the results of the code audit conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered neither "endorsement" nor "disapproval" of the guarantee of the correct business model of the analyzed project, nor as guarantee on the operation or viability of the implemented financial product.

Red4Sec makes full effort and applies every resource available for each audit, however it does not warrant the function, nor the safety of the project and it cannot be deemed a sufficient assessment of the code's utility and safety, bug-free status, or any other declarations of the project. Additionally, Red4Sec makes no security assessments or judgments about the underlying business strategy, or the individuals involved in the project.

Blockchain technology and cryptographic assets come with its own new risks and challenges, where the ecosystem, platform, its programming language, and other software related to said technology can have vulnerabilities that could lead to exploits. As a result, the audit cannot guarantee the explicit security of the audited projects.

The audit reports can be used to improve the code quality of smart contracts, to help limit the vectors of attack and to lower the high level of risks associated with utilizing new and continually changing technologies such as cryptographic tokens and blockchain, but they are unable to detect any future security concerns with the related technologies.

Scope

Red4Sec Cybersecurity has made a thorough audit of the **Genome Mining PR #49** security level against attacks, identifying possible errors in the design, configuration or programming; therefore, guaranteeing the availability, integrity and confidentiality of the project and the possible assets treated and stored.

The scope of this evaluation includes the following items provided by **Altered State Machine**:

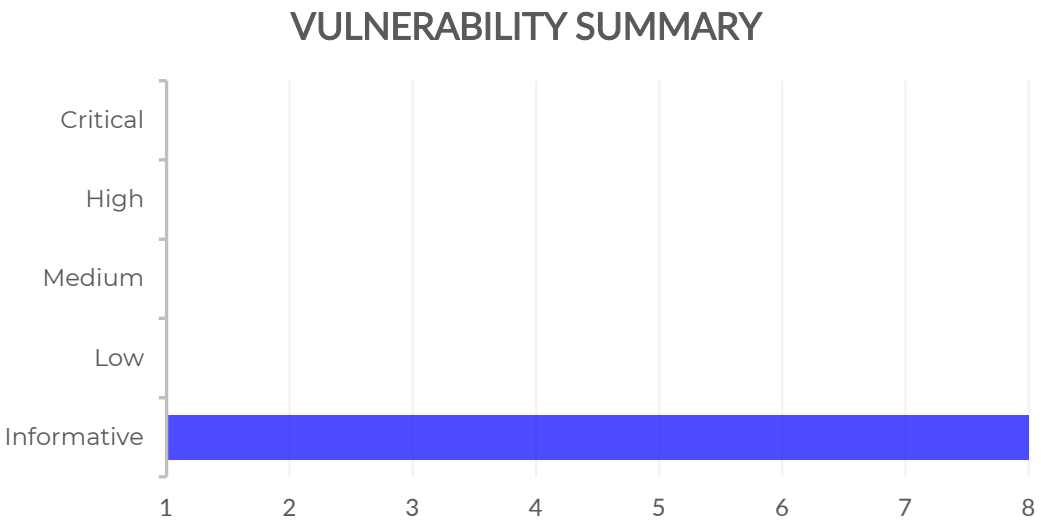
- https://github.com/altered-state-machine/genome-mining-contracts/pull/49/
  - `commit: 725124ff0a688a2d18eaf09b96699396e3814643`

Executive Summary

The security audit against **Genome Mining PR #49** has been conducted between the following dates: **07/25/2022** and **08/01/2022**.

Once the analysis of the technical aspects has been completed, the performed analysis shows that the audited source code contains non-critical issues that were mitigated promptly.

During the analysis, a total of **8 issues** were detected, these vulnerabilities have been classified by the following level of risks, defined in Annex A.



Conclusions

To this date, **08/03/22**, the conclusion resulting from the conducted review of the pull request from **GenII brain contracts** is that it can be considered safe since the auditors have not found any known vulnerabilities.

Nevertheless, Red4Sec has found a few potential improvements, these do not pose any risk by themselves, and we have classified such issues as informative only, but they will help to continue to improve the security and quality of its developments.

The general conclusions of the performed audit are:

- The **absence of the Unit Test** was detected, during the security review, this is a highly recommended practice that has become mandatory in projects destined to manage large amounts of capital.
- A few low impact issues were detected and classified only as informative, but they will continue to help Genome Mining to improve the security and quality of its developments.
- Certain methods **did not make the necessary input checks** in order to guarantee the integrity and expected arguments format.
- It was informed that although the contract has functions to resign to the admin privileges throughout the life of the contract, this decision initially relies with the project itself.
- All the proposed recommendations that were considered necessary by Red4Sec in order to improve the security of the project were properly applied by the **Altered State Machine** team.

Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered upon the security audit.

List of vulnerabilities

Below, we have gathered a complete list of the vulnerabilities detected by Red4Sec, presented and summarized in a way that can be used for risk management and mitigation.

Table of vulnerabilities			
ID	**Vulnerability**	**Risk**	**State**
GBC-01	Absence of unit tests	**Informative**	**Fixed**
GBC-02	Lack of inputs validation	**Informative**	**Fixed**
GBC-03	Improve decentralization	**Informative**	**Assumed**
GBC-04	GAS optimization	**Informative**	**Fixed**
GBC-05	Pragma inconsistency	**Informative**	**Fixed**
GBC-06	Consider the use of _mint or _safeMint	**Informative**	**Closed**
GBC-07	Unnecessary update roles methods	**Informative**	**Fixed**
GBC-08	tokenURI execution cost	**Informative**	**Assumed**

Vulnerability details

In this section, we provide the details of each of the detected vulnerabilities indicating the following aspects:

- Category
- Active
- Risk
- Description
- Recommendations

Absence of unit tests

Identifier	Category	Risk	State
GBC-01	Testing and Documentation	**Informative**	**Fixed**

The absence of certain Unit Test has been detected, during the pull request security review, although the project does include Unit Tests for most of the functions, the changes introduced by this pull request are not included. This is a highly recommended practice that has become mandatory in projects destined to manage large amounts of capital.

The audit team has developed certain tests for critical functions, such as the `base58` implementation. However, it is recommended to include tests for all the functionalities of the project.

```
Test #97
  Random Buffer: <Buffer eb b4 69 f5 a4 17 e4 a9 e9 dc 8b 78 f3 7c 29 9a 48 c5 03 37 f8 fa b0 b1 73 1d a8 0f 37>
  Contract result: 0x4268636e45475831373653424c487435696d6d6b484243717841485 06f324e547362525a72483461
  JS result: 4268636e45475831373653424c487435696d6d6b484243717841485 06f324e547362525a72483461
  Match ✓
Test #98
  Random Buffer: <Buffer f1 61 1a da ce>
  Contract result: 0x55455734727475
  JS result: 55455734727475
  Match ✓
Test #99
  Random Buffer: <Buffer 06 27 16 e0 fa df 04 52 b8 5b 7e ff 54 1d 15 cf>
  Contract result: 0x6d34715352486871753445 6b655a48563266653155
  JS result: 6d34715352486871753445 6b655a48563266653155
  Match ✓
    ✓ Test base58 implementation (12430ms)
```

For the safety development of any project unitary tests are essential, and its periodical execution is fundamental.

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/altered-state-machine/genome-mining-contracts/pull/49/commits/cd37b10e4e967f642253f8d380b0538772913864

Lack of inputs validation

Identifier	Category	Risk	State
GBC-02	Data Validation	**Informative**	**Fixed**

Certain methods of the different contracts in the `Genome Mining` project do not properly check the arguments, which can lead to major errors.

Below we list the most significant examples.

During the execution of the following methods, it is not verified that the provided addresses sent by arguments are valid, therefore it allows `address(0)`.

- ASMBrainsGenII.sol#L20
- ASMBrainsGenII.sol#L30
- ASMBrainsGenII.sol#L95
- ASMBrainsGenIIMinter.sol#L37-L40
- ASMBrainsGenIIMinter.sol#L144
- ASMBrainsGenIIMinter.sol#L154
- ASMBrainsGenIIMinter.sol#L164

Fixes Review

This issue has been addressed in the following commits:

- https://github.com/altered-state-machine/genome-mining-contracts/pull/49/commits/cd37b10e4e967f642253f8d380b0538772913864
- https://github.com/altered-state-machine/genome-mining-contracts/commit/c88cb0fb893044ff008e41c193643b948a09db6d

Improve decentralization

Identifier	Category	Risk	State
GBC-03	Governance	**Informative**	**Assumed**

In order to promote transparency, it would be advisable to improve the logic of various administrative methods in the `Genome Mining` contracts.

It is important to mention that the governance of the contract has the possibility of updating certain values of the contract, reducing part of the concept of decentralized trust.

A few of the administrative functionalities that are under the control of the team and have the ability to alter the logic of the operations of the contracts are; `updateBaseURI`, `updateConfiguration`, `updateConverter` and `updateBrain`.

There are certain good practices that help mitigate this problem, such as; allowing to establish these values only once or the use of TimeLock, to start the update operations, emit events when the update operation is requested and finally issue a last notification and execute the update operation after the TimeLock is completed.

Recommendations

- Reduce the number of administrative methods by lowering the number of values that can alter the logic of the contract.
- Implement TimeLocks to improve the transparency of these operations.

Source Code References

- ASMBrainsGenII.sol#L68
- ASMBrainsGenIIMinter.sol#L134
- ASMBrainsGenIIMinter.sol#L154
- ASMBrainsGenIIMinter.sol#L165

Fixes Review

ASM Notes: *We will use Gnosis Safe for administrative methods and will adopt the TimeLock contract soon in the near future.*

GAS optimization

Identifier	Category	Risk	State
GBC-04	Codebase Quality	**Informative**	**Fixed**

Software optimization is the process of modifying a software system to make an aspect of it work more efficiently or use less resources. This premise must be applied to smart contracts as well, so that they execute faster or in order to save GAS.

On EVM based blockchains, GAS is an execution fee which is used to reward miners for the computational resources required to power smart contracts. If the network usage is increasing, so will the value of GAS optimization.

These are some of the requirements that must be met to reduce GAS consumption:

- Short-circuiting.
- Remove redundant or dead code.
- Delete unnecessary libraries.
- Explicit function visibility.
- Use of proper data types.
- Use hard-coded CONSTANT instead of state variables.
- Avoid expensive operations in a loop.
- Pay special attention to arithmetical operations and comparisons.

Increase operation optimization

During the audit, it was found that it is possible to optimize all the for loops in the project. There are two main ways to increment/decrement variables in solidity:

```
--i/++i will decrement/increment the value of i, and then return the decremented/
incremented value.

i--/i++ will decrement/increment the value of i, but return the original value that i he
ld before being decremented/incremented.
```

Since the return of the decrement/increment operation of the for loop is indifferent and discarded, both `i--/i++` or `--i/++i` instructions are completely valid instructions, however, the `--i/++i` instruction has a considerably lower cost compared to decrementing/incrementing a variable using `i--/i++`, for this reason it is convenient to use `--i/++i` for the decrements/increments of the for loops.

An example of this behaviour can be seen below:

```
for (uint256 i = 0; i < data_.length; i++) {
    m = int256(size - 1);
    for (carry = uint8(data_[i]); m > high || carry != 0; m--) {
        carry = carry + 256 * uint8(slot[uint256(m)]);
        slot[uint256(m)] = bytes1(uint8(carry % 58));
        carry /= 58;
    }
    high = m;
}
```

References

- https://docs.soliditylang.org/en/latest/types.html#compound-and-increment-decrement-operators

Source references

- contracts/libraries/IPFS.sol#L30
- contracts/libraries/IPFS.sol#L32
- contracts/libraries/IPFS.sol#L40
- contracts/libraries/IPFS.sol#L43
- contracts/libraries/IPFS.sol#L59

Unnecessary Method

During the audit, the existence of the `currentTime` method of the `ASMBrainsGenIIMinter` contract was detected. This method is unnecessary since it only returns the result of `block.timestamp` and can implement its logic where necessary. Additionally, the execution of this type of code consumes unnecessary GAS during deployment.

Source reference

- ASMBrainsGenIIMinter.sol#L173-L176

Wrong Visibility

In order to simplify the contract for the users, it is recommended to turn the following variables to `private`.

```
contract IPFS {
    bytes public constant ALPHABET = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";
```

Source reference

- contracts/libraries/IPFS.sol#L10

Optimize require messages

Ethereum Virtual Machine operates under a 32-byte word memory model where an additional gas cost is paid by any operation that expands the memory that is in use.

Therefore, exceeding error messages of this length means increasing the number of slots necessary to process the require, reducing the error messages to 32 bytes or less would lead to saving gas.

Source reference

- contracts/helpers/Util.sol#L12
- contracts/helpers/Util.sol#L20
- contracts/helpers/Util.sol#L29

Use custom errors instead of require

Solidity **0.8.4** introduced custom errors, a more efficient way to notify users of an operation failure.

It has been verified that the `ASMBrainsGenII` and `ASMBrainsGenIIMinter` contracts are managing the error messages through the `require` command, this instruction is more expensive than the use of custom errors because it requires placing the error message on the stack, whether or not it is reverted the transaction, in addition to this, is more susceptible to using long error messages that will produce a higher cost of gas, whatever the result of the condition.

```
require(quantity > 0, "Hashes cannot be empty");
require(quantity <= config.maxQuantityPerTx, "Too many hashes");
require(currentTime() >= config.startTime, "Not started");
require(currentTime() < config.endTime, "Already finished");
// Only allow use enery accumulated from previous production cycles
require(periodId < energyConverter.getCurrentPeriodId(), "Invalid periodId");
require(_verify(_hash(hashes, msg.sender, brain.numberMinted(msg.sender)), signature), "Invalid signature");
```

```
require(quantity <= remainingSupply(periodId), "Max supply exceeded");
PeriodConfig memory config = configuration[periodId];
require(quantity > 0, "Hashes cannot be empty");
require(quantity <= config.maxQuantityPerTx, "Too many hashes");
require(currentTime() >= config.startTime, "Not started");
require(currentTime() < config.endTime, "Already finished");
// Only allow use enery accumulated from previous production cycles
require(periodId < energyConverter.getCurrentPeriodId(), "Invalid periodId");
require(_verify(_hash(hashes, msg.sender, brain.numberMinted(msg.sender)), signature), "Invalid signature");
```

Custom errors are more gas efficient than revert strings in terms of deployment and runtime cost when the revert condition is met. In order to save gas, it is recommended use custom errors instead of revert strings.

The new custom errors are defined through the `error` statement, and they can also receive arguments. As indicated in the following example:

```
error Unauthorized();
error InsufficientPrice(uint256 available, uint256 required);
```

Afterwards, it is enough to just call them when needed using `if` conditionals:

```
if (msg.sender != owner) revert Unauthorized();
if (amount > balance[msg.sender]) {
            revert InsufficientPrice({
                available: balance[msg.sender],
                required: amount
            });
}
```

This behavior has been observed in:

- ASMBrainsGenII.sol#L49
- ASMBrainsGenIIMinter.sol#L100-L112

Reference

- https://blog.soliditylang.org/2021/04/21/custom-errors

Caching arrays length in loops

Everytime an array iteration occurs in a loop; the solidity compiler will read the array's length. Thus, this is an additional `sload` operation for storage arrays (100 more extra gas for each iteration except for the first) and an additional `mload` action for memory arrays (3 additional gas for each iteration except for the first).

```
for (uint i; i < array.length; ++i) {
    // ...
}
```

The array length (in stack) might be cached to reduce these additional costs:

```
uint length = array.length;
for (uint i; i < length; ++i) {
    // ...
}
```

The `sload` or `mload` action is only performed once in the example above before being replaced by the inexpensive `dupX` instruction.

Source reference

- contracts/libraries/IPFS.sol#L30
- contracts/libraries/IPFS.sol#L59

Fixes Review

This issue has been addressed in the following commits:

- https://github.com/altered-state-machine/genome-mining-contracts/pull/49/commits/cd37b10e4e967f642253f8d380b0538772913864
- https://github.com/altered-state-machine/genome-mining-contracts/commit/c88cb0fb893044ff008e41c193643b948a09db6d

Pragma inconsistency

Identifier	Category	Risk	State
GBC-05	Outdated Software	**Informative**	**Fixed**

During the security review a small inconsistency has been found between the compiler versions used in the different files of the project.

The majority of the audited files establish the `^0.8.15` version of Solidity in the pragma as indicated in `foundry.toml` as well.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.15;
```

However, some of the dependencies used for the project do not use the same version, it is recommended to always use the same compiler version for all the contracts used, and it is always a good policy to use the most up to date version of the pragma .

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;
```

Recommendations

- It is recommended to use the same version for all the dependencies used throughout the project.

References

- https://github.com/ethereum/solidity/blob/develop/Changelog.md

Source Code References

- contracts/interfaces/IConverter.sol#L3
- contracts/helpers/Util.sol#L3

Fixes Review

ASM Notes: *Solidity compiler versions are changed to `0.8.13` to be aligned with other Genome Mining contracts, as we are sharing the same codebase with previously deployed contracts.*

This issue has been addressed in the following commit:

- https://github.com/altered-state-machine/genome-mining-contracts/pull/49/commits/cd37b10e4e967f642253f8d380b0538772913864

Consider the use of _mint or _safeMint

Identifier	Category	Risk	State
GBC-06	Business Logic	**Informative**	Closed

This issue should be received simply as a note for the ASM team to study which functionality best suits their needs. By inheriting the functionalities from `ERC721A` it is possible to use `_mint` or `_safeMint` during the mint of the tokens.

A good reference to decide which is more suitable would be the following:

- *If the PROJECT is paying for the minting of tokens, use `_mint`; the `_safeMint` might cost you an arbitrary amount of money due to choices made by the recipient of the tokens. This is enough to deter you from considering it.*
- *If the USERS are paying for the minting of tokens and you expect buyers to be composing functionalities with smart contracts, use `_safeMint`. There is some marginal benefit of allowing the extra features with smart contracts this allows.*
- *If the USERS are paying and you expect INDIVIDUAL PEOPLE to buy tokens, then use `_mint`. The extra features in `_safeMint` are not expected.*

The extra cost from using `_safeMint` is non-zero and cost is always a concern.

Recommendations

- Study which of the two scenarios adjusts better to the business logic of the project.

References

- https://ethereum.stackexchange.com/questions/115280/mint-vs-safemint-which-is-best-for-erc721/115293#115293

Source Code References

- ASMBrainsGenII.sol#L39

Fixes Review

ASM Notes: *We are using `_mint` because it's for users to mint the GenII Brains NFTs and we are only expecting EOA.*

Unnecessary update roles methods

Identifier	Category	Risk	State
GBC-07	Codebase Quality	**Informative**	**Fixed**

During the audit of the `ASMBrainsGenII` contract, the existence of certain unnecessary methods have been identified that can be eliminated, improving the readability of the code and reducing the GAS cost of the deployment of the smart contract.

The `updateMinter` and `updateAdmin` methods of the `ASMBrainsGenII` contract are redundant and therefore unnecessary, since the following native methods exist: `AccessControl`, `revokeRole` and `grantRole`.

```
function updateAdmin(address _oldAdmin, address _newAdmin) external onlyRole(ADMIN_ROLE) {
    _revokeRole(ADMIN_ROLE, _oldAdmin);
    _grantRole(ADMIN_ROLE, _newAdmin);
}
```

In `updateAdmin`, the private methods `_revokeRole` and `_grantRole` of the `AccessControl` are invoked to make the necessary modifications, however, it is possible to directly use the public methods `revokeRole` and `grantRole`, in addition to `renounceRole` if necessary.

Additionally, the term "update" can lead to confusion since the logic of these methods does not guarantee that the account with the previous roles will be deleted if it is not correctly specified in the arguments. There is no control that verifies that the previous values of `_oldMinter` or `_oldAdmin` actually run through argument, being able to bypass the logic that would revoke the role of the previous minter/admin.

Likewise, the existence of multiple minters or admins simultaneously is not prevented through the use of the `grantRole` function or losing the role through `revokeRole` or `renounceRole`.

Recommendations

- It is recommended to remove all the unnecessary methods in order to improve code readability and optimize the contract during deploy.

Source Code References

- ASMBrainsGenII.sol#L79
- ASMBrainsGenII.sol#L95
- AccessControl.sol#L144-L161

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/altered-state-machine/genome-mining-contracts/commit/c88cb0fb893044ff008e41c193643b948a09db6d

tokenURI execution cost

Identifier	Category	Risk	State
GBC-08	Business Logic	**Informative**	**Assumed**

It is important to mention that the cost of the `tokenURI` method could be too high in certain circumstances and the ASM team should study this particular case.

The `tokenURI` method of the `ASMBrainsGenII` contract uses the `cidv0` and `base58` methods inherited from `IPFS`, which have a high computational cost and should be used only as query methods. Therefore, it is convenient to check if the project needs to call `tokenURI` from another smart contract in the future.

Source Code References

- contracts/libraries/IPFS.sol#L55
- contracts/ASMBrainsGenII.sol#L48
- contracts/libraries/IPFS.sol#L15

Fixes Review

ASM Notes: *IPFS hashes will be stored in `bytes32` to save the gas for users when they mint the GenII brain tokens. `tokenURI` is only used in query methods so it will not cause any gas-related issues, and we don't have a plan to use it in another smart contract in the future.*

Annexes

Annex A – Vulnerabilities Severity

Red4Sec determines the vulnerabilities severity in the following levels of risk according to the impact level defined by CVSS v3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST):

Vulnerability Severity

The risk classification has been made on the following 5 value scale:

Severity	Description
Critical	Vulnerabilities that possess the highest impact over the systems, services and/or sensitive information. The existence of these vulnerabilities is dangerous and should be fixed as soon as possible.
High	Vulnerabilities that could compromise severely compromise the service or the information it manages even if the vulnerability requires expertise to be exploited.
Medium	Vulnerabilities that on their own can have a limited impact and/or that combined with other vulnerabilities could have a greater impact.
Low	These vulnerabilities do not suppose a real risk for the systems. Also includes vulnerabilities which are extremely hard to exploit or whose impact on the service is low.
Informative	It covers various characteristics, information or behaviours that can be considered as inappropriate, without being considered as vulnerabilities by themselves.

Invest in Security, invest in your future