HashEx
AUDITS

Halloween Finance Audit

21/4/2022

Disclaimer

HashEx reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Spade to perform a security review.

HashEx Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

HashEx Reports should not be used in any way to make decisions around investment or involvement with any particular project.These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

HashEx Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. HashEx’s position is that each company and individual are responsible for their own due diligence and continuous security. HashEx’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a HashEx report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to HashEx by a Client.

- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.

- Representation that a Client of HashEx has indeed completed a round of auditing with the intention to increase the quality of the company/ product’s IT infrastructure and or source code.

OVERVIEW

Project Summary

Project Name	Halloween Finance
Description	DeFi
Platform	Aurora Network
2ombtoken	https://github.com/halloweenAurora/halloweenfinance/blob/master/contracts/Tomb.sol
masonary	https://github.com/halloweenAurora/halloweenfinance/blob/master/contracts/Masonry.sol
treasury	https://github.com/halloweenAurora/halloweenfinance/blob/master/contracts/Treasury.sol
oracle	https://github.com/halloweenAurora/halloweenfinance/blob/master/contracts/Oracle.sol
Two_ombGenesisRewardPool	https://github.com/halloweenAurora/halloweenfinance/blob/master/contracts/distribution/TombGenesisRewardPool.sol

Audit Summary

Delivery Date	April 21th, 2022
Method of Audit	mythril tools for analyzing, Manual Review
Timeline	Story Points - 64

Vulnerability Summary

Total Issues	0
Total Critical	0
Total High	0
Total Medium	0
Total Low	0
Total Informational	0

Executive Summary

Our detailed audit methodology was as follows:

Step 1
A manual line-by-line code review to ensure the logic behind each function is sound and safe from common attack vectors.
Step 2
Simulation of hundreds of thousands of Smart Contract Interactions on a test blockchain using a combination of automated test tools and manual testing to determine if any security vulnerabilities exist.
Step 3
Consultation with the project team on the audit report pre-publication to implement recommendations and resolve any outstanding issues.

Grading

The following grading structure was used to assess the level of vulnerability found within all Smart Contracts:

Threat Level	Definition
Critical	Severe vulnerabilities which compromise the entire protocol and could result in immediate data manipulation or asset loss.
High	Significant vulnerabilities which compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Medium	Vulnerabilities which if not fixed within in a set timescale could compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Low	Low level vulnerabilities which may or may not have an impact on the optimal performance of the Smart contract.
Informational	Issues related to coding best practice which do not have any impact on the functionality of the Smart Contracts.

Automation Testing

The audit proceess involves:

1. using mythril tools for analyzing the exploit in the code → None found

2. Manual line by line reviewing of code to check for logical/calculation error → no issues found.

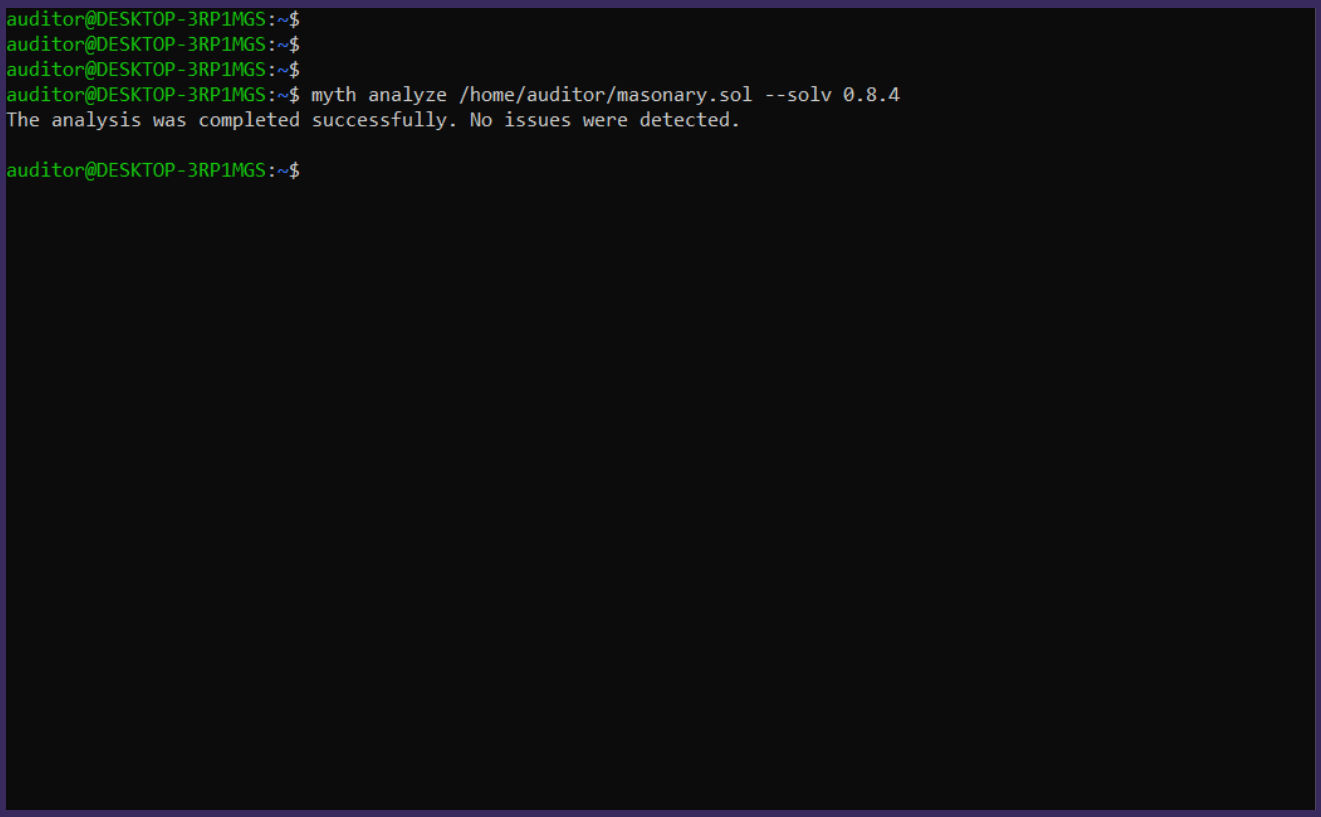
File: 2ombToken.sol

```
auditor@DESKTOP-3RP1MGS:~$ nano 2ombToken.sol
auditor@DESKTOP-3RP1MGS:~$ myth analyze /home/auditor/2ombToken.sol --solv 0.8.4
The analysis was completed successfully. No issues were detected.

auditor@DESKTOP-3RP1MGS:~$
```

No issues found.

File: Masonary.sol



No issues found.

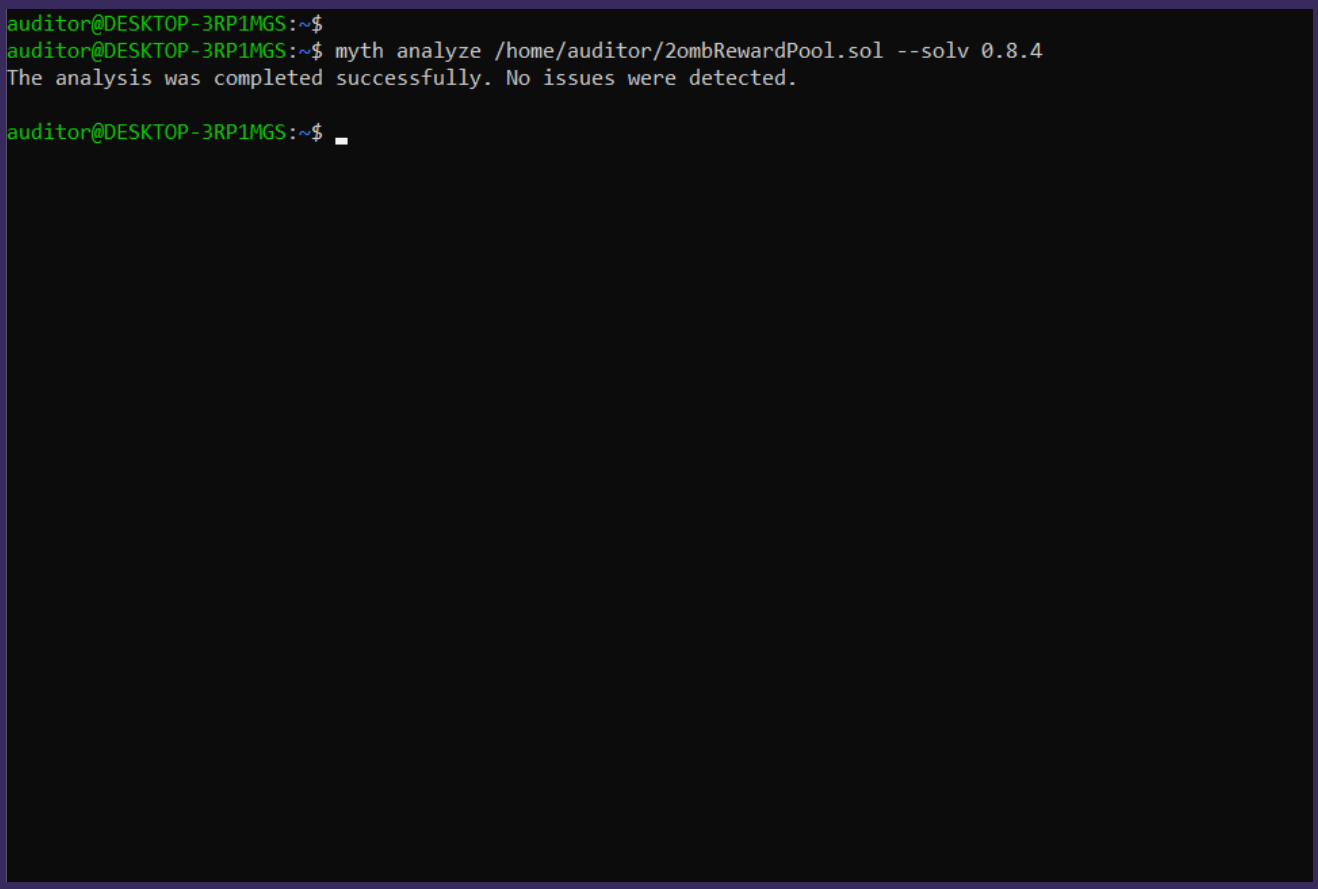
File : Treasury.sol

```
auditor@DESKTOP-3RP1MGS:~$
auditor@DESKTOP-3RP1MGS:~$ myth analyze /home/auditor/treasury.sol --solv 0.8.4
The analysis was completed successfully. No issues were detected.

auditor@DESKTOP-3RP1MGS:~$
```

No issues found.

File: 2ombRewardPool.sol



No issues found.

Best Practices

1.Hardcoded addresses should be checked carefully before deployment.
2.Ownership transfer function - It is good practice to implement an acceptOwnership style to prevent ownership sent to invalid addresses by human error. Code flow similar to below.

```
function transferOwnership(address _newOwner) public
onlyOwner

{

newOwner = _newOwner;

}

function acceptOwnership() public

{

require(msg.sender == newOwner, “Not authorized”);
emit OwnershipTransfered(owner,newOwner);
owner = newOwner;
newOwner = address(0);

}
```

Appendix

Finding Categories

Gas Optimization
Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations
Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue
Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow
Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code
Volatile Code findings refer to segments of code that behave unexpectely on certain edge cases that may result in avulnerability.

Data Flow
Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a structassignment operation affecting an in-memory struct rather than an instorage one.

Language Specific
Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

Coding Style
Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency
Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers
Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error
Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code
Code that otherwise does not affect the functionality of the codebase and can be safely omitted.