AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

TABLE OF CONTENTS

contact@auditmos.com

AUDITMOS

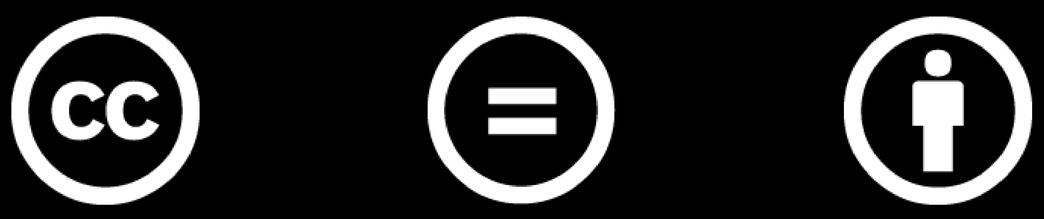
Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

LICENSE



contact@auditmos.com

AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

DISCLAIMER

contact@auditmos.com

AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

APPROACH AND METHODOLOGY

PURPOSE

1. Determine the correct operation of the protocol, according to the design specification.
2. Identify possible vulnerabilities that could be exploited by an attacker.
3. Detect errors in the smart contract that could lead to unexpected behavior.
4. Analyze whether best practices were followed during development.
5. Make recommendations to improve security and code readability.

CODEBASE

Repository	https://github.com/GameSwift/gs-evm-contracts/tree/develop
Branch	develop
Commit hash	ffe4aba0b3574213b6dd1a9ef8c9cc4fbfcb12aa

METHODOLOGY

1. Reading the available documentation and understanding the code.
2. Doing automated code analysis and reviewing dependencies.
3. Checking manually source code line by line for security vulnerabilities.
4. Following guildlines and recommendations.
5. Preparing this report.

contact@auditmos.com

AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

DESCRIPTION

Issues Categories:

Severity	Description
CRITICAL	vulnerability that can lead to loss of funds, failure to recover blocked funds, or catastrophic denial of service.
HIGH	vulnerability that can lead to incorrect contract state or unpredictable operation of the contract.
MEDIUM	failure to adhere to best practices, incorrect usage of primitives, without major impact on security.
LOW	recommendations or potential optimizations which can lead to better user experience or readability.

Each issue can be in the following state:

State	Description
PENDING	still waiting for resolving
ACKNOWLEDGED	know but not planned to resolve for some reasons
RESOLVED	fixed and deployed

contact@auditmos.com

AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

AUDIT SCOPE

1. getting to know the project	✓
2. research into architecture	✓
3. manual code read	✓
4. permissions of state changing functions	✓
5. identify common Solidity vulnerabilities	✓
6. test coverage	✓
7. static analysis	✓
8. storage key overlaps	✓
9. DOS possibilities by malicious attacker	✓
10. steal funds possibilities	✓

contact@auditmos.com

AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

FINDINGS

Finding	Severity	Status
#1 - Excess fees are not returned by the release() function	HIGH	RESOLVED
#2 - Unclaimed tokens are locked in the Vesting.sol contract	HIGH	RESOLVED
#3 - Emit event in crucial places	LOW	ACKNOWLEDGED
#4 - Add NatSpec documentation	LOW	ACKNOWLEDGED

contact@auditmos.com

AUDITMOS

Secure your space

GAMESWIFT

audit / code review report

June 23, 2023

#1 - EXCESS FEES ARE NOT RETURNED BY THE RELEASE() FUNCTION

Severity	Status
HIGH	RESOLVED

The main impact is of a financial nature since the existing implementation does not refund any additional Ether (msg.value) paid above the ethFee requirement. Users who surpass the required fee will not have their extra payment returned, resulting in an unintentional loss of funds.

Within the release() function, users are obligated to pay a fee (ethFee) in order to initiate the token release process.

Nevertheless, the function currently lacks a mechanism to handle instances where users pay an amount (msg.value) that exceeds the required fee. Presently, if msg.value > ethFee, the contract proceeds to accept the payment without returning the surplus amount to the user.

RECOMMENDATION

Return the difference to the user

```
uint256 dust = msg.value - ethFee;
(bool sent, ) = address(msg.sender).call{value: dust}("");
require(sent, "Failed to send Ether");
```

contact@auditmos.com

GAMESWIFT

audit / code review report

June 23, 2023

#2 - UNCLAIMED TOKENS ARE LOCKED IN THE VESTING.SOL CONTRACT

Severity	Status
HIGH	RESOLVED

Unclaimable tokens are stuck in the Vesting.sol contract.

This issue leads to tokens that are not claimed within the specified timeframe, as well as "dust" tokens resulting from minor calculation discrepancies, being trapped indefinitely within the Vesting.sol contract. Consequently, these tokens are effectively lost, reducing the overall available supply and potentially causing distortions in the market dynamics of the token.
The problem is that there might be the scenario where some tokens are left unclaimed (for example, if no user claim rewards at all) OR there is some dust left.
In this case, the contract owners might want to retrieve the unclaimed tokens.

RECOMMENDATION

Implement a token recovery function that can only be executed by addresses with the dedicated role, but with the condition that it can only be performed after the conclusion of the vesting period. This functionality will enable addresses possessing the specified role to retrieve any surplus tokens from the contract.

```
function recoverERC20(uint256 tokenAmount) external only managerRole {
        IERC20(token).safeTransferFrom(address(this), msg.sender, tokenAmount);
 }
```

#3 - EMIT EVENT IN CRUCIAL PLACES

Severity	Status
LOW	ACKNOWLEDGED

Emitting events serves as an essential tool for transparency and information flow within the contract.

By emitting events, you provide an immutable record of state changes, allowing external systems and users to easily track and react to these changes. This promotes trust and visibility, enhancing the overall security and auditability of the smart contract.

When emitting events, consider including relevant information about the state change, such as the affected addresses, updated values, and any additional contextual data. This helps external applications to accurately interpret and respond to the emitted events.

Remember to define and document your event structures clearly, ensuring they align with the purpose and requirements of the contract. Consistency in event naming conventions and event parameter definitions across the contract can aid in readability and understanding.

By incorporating event emission into functions where state changes occur, you enhance the transparency and integrity of your smart contract, making it more robust and secure.

#4 - ADD NATSPEC DOCUMENTATION

Severity	Status
LOW	ACKNOWLEDGED

When working with Solidity smart contracts, utilizing natspec documentation is highly recommended as it improves the readability, maintainability, and overall security of the contract.

RECOMMENDATION

Effectively using natspec:

1. Document contract functionality: Begin by providing a high-level overview of the contract's purpose, functionality, and main components. Explain the contract's role within the system or application it serves.
2. Document function behavior: For each function, document its purpose, expected inputs, outputs, and any events emitted. Clearly specify the function's intended behavior, including any constraints, limitations, or side effects. Consider including examples or scenarios to illustrate how the function should be used.
3. Clarify input requirements: Document the expected format, type, and range of input parameters for each function. Specify any preconditions or validation rules that should be followed to ensure the function executes correctly. Clearly define any potential risks or security considerations associated with specific input values.
4. Explain return values: Describe the expected return values of functions and their meanings. Document any potential error codes, error conditions, or exceptional scenarios that users should be aware of when interpreting return values.
5. Document events: Clearly define the purpose and structure of emitted events. Explain the significance of each event parameter and provide guidelines for interpreting the event data. Consider including examples or use cases that demonstrate how events can be utilized.
6. Address security considerations: Use natspec documentation to highlight potential security risks, attack vectors, or best practices relevant to the contract. Provide guidance on how to mitigate these risks and implement secure coding practices.
7. Maintain consistency and clarity: Follow a consistent style and format throughout your natspec documentation to ensure readability and ease of understanding. Use clear and concise language, avoid jargon, and provide relevant examples or illustrations whenever possible.

auditmos.com

AUDITMOS

Secure your space

contact@auditmos.com