AUDITMOS

Secure your space

PLAYNITY

audit / code review report

December 15, 2021

TABLE OF CONTENTS

contact@auditmos.com

AUDITMOS

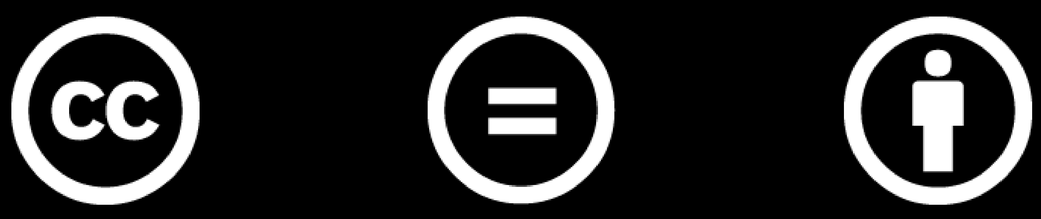
Secure your space

PLAYNITY

audit / code review report

December 15, 2021

LICENSE



contact@auditmos.com

AUDITMOS

Secure your space

PLAYNITY

audit / code review report

December 15, 2021

DISCLAIMER

contact@auditmos.com

AUDITMOS

Secure your space

PLAYNITY

audit / code review report

December 15, 2021

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://gitlab.com/playnity/contracts-evm/
Branch	develop
Commit hash	319deb3aaf86111dfda52cdd3c9e38c53fd14bd7

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.

AUDITMOS

Secure your space

PLAYNITY

audit / code review report

December 15, 2021

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

PLAYNITY

audit / code review report

December 15, 2021

FINDINGS

Finding	Severity	Status
#1 - SafeMath not obligatory in Solidity v0.8+	LOW	ACKNOWLEDGED
#2 - Use reentrancy guard in more efficient way	LOW	ACKNOWLEDGED
#3 - Check return values of erc20 transfer and transferFrom	LOW	ACKNOWLEDGED
#4 - Additional check for feesTransferAddress	LOW	ACKNOWLEDGED
#5 - Emit events before external calls	LOW	ACKNOWLEDGED
#6 - Recommendations for gas optimisation	LOW	ACKNOWLEDGED

contact@auditmos.com

AUDITMOS

Secure your space

PLAYNITY

audit / code review report

December 15, 2021

#1 - SAFEMATH NOT OBLIGATORY IN SOLIDITY V0.8+

Severity	Status
LOW	ACKNOWLEDGED

RECOMMENDATION

In solidity v0.8+ you don’t have to use SafeMath as it is given by default

PROOF OF SOURCE

https://gitlab.com/playnity/contracts-evm/-/blob/319deb3aaf86111dfda52cdd3c9e38c53fd14bd7/contracts/Distribution.sol#L3

contact@auditmos.com

PLAYNITY

audit / code review report

December 15, 2021

#2 - USE REENTRANCY GUARD IN MORE EFFICIENT WAY

Severity	Status
LOW	ACKNOWLEDGED

In Staking.sol reentrancy guard is used in public functions like withdraw, stake, compoundReward etc, but then these functions are called from methods which are not secured for reentrancy. It is recommended to split implementation of these functions in Staking.sol.

RECOMMENDATION

Consider to change:

```
function withdraw(uint256 amount)
  public
  virtual
  nonReentrant
{
    return _withdraw(amount);
}
```

and move implementation original withdraw into a new internal function:

```
function _withdraw(uint256 amount)
  internal
  virtual
  updateReward(msg.sender)
{
// the original implementation.
}
```

Then in child contracts you can just add ‘nonReentrant’ to public functions and call internal versions of original ones.

PROOF OF SOURCE

https://gitlab.com/playnity/contracts-evm/-/blob/319deb3aaf86111dfda52cdd3c9e38c53fd14bd7/contracts/Staking.sol#L111

PLAYNITY

audit / code review report

December 15, 2021

#3 - CHECK RETURN VALUES OF ERC20 TRANSFER AND TRANSFERFROM

Severity	Status
LOW	ACKNOWLEDGED

Check return values of ERC20 transfer and transferFrom. In general most of ERC20 implementations are fail proof and these functions always returns true, but for some very specific implementations it might not be the case as well as for malicious contracts.

RECOMMENDATION

Consider to change:

a) Distribution.sol line 52. ‘token.transfer’ return value is not checked.
b) Staking.sol lines (107, 126, 128, 138, 231, 232)
c) StakingBonded.sol (100, 123, 132, 143)
d) Vesting.sol (77, 150)

PROOF OF SOURCE

Described above.

AUDITMOS

Secure your space

PLAYNITY

audit / code review report

December 15, 2021

#4 - ADDITIONAL CHECK FOR FEESTRANSFERADDRESS

In StakingBonded.sol it might be better to check if feesTransferAddress is correct that means if it is regular address or a contract which supports withdrawing from it's balance.

Severity	Status
LOW	ACKNOWLEDGED

PROOF OF SOURCE

https://gitlab.com/playnity/contracts-evm/-/blob/319deb3aaf86111dfda52cdd3c9e38c53fd14bd7/contracts/StakingBonded.sol#L100

contact@auditmos.com

PLAYNITY

audit / code review report

December 15, 2021

#5 - EMIT EVENTS BEFORE EXTERNAL CALLS

Severity	Status
LOW	ACKNOWLEDGED

Generally a recommended solution would be to emit events before external calls (in case of reentrancy and event driven logic it may cause an issues).

RECOMMENDATION

Consider to change:

a) Staking.sol
- line 138 rewardsToken.transfer(msg.sender, reward);
- line 139 emit RewardPaid(msg.sender, reward);

b) Staking.sol
- line 128 stakingToken.transfer(msg.sender, amount.sub(fees)); and
- line 129 emit Withdrawn(msg.sender, amount);

c) StakingBonded.sol
- line 123 stakingToken.transfer(msg.sender, amount);
- line 124 emit Withdrawn(msg.sender, amount);

d) Vesting.sol
- line 77 token.transfer(msg.sender, amount);
- line 78 emit Withdrawn(msg.sender, amount);

PROOF OF SOURCE

Described above.

PLAYNITY

audit / code review report

December 15, 2021

#6 - RECOMMENDATIONS FOR GAS OPTIMISATION

Some of best practices are described below.

Severity	Status
LOW	ACKNOWLEDGED

RECOMMENDATION

a) In Staking.sol unstakingFee can be constant

b) Some functions should be declared as external (as they are not used internally):

- Token.mint,
- StakingBonded.unbondingSummary

PROOF OF SOURCE

https://gitlab.com/playnity/contracts-evm/-/blob/319deb3aaf86111dfda52cdd3c9e38c53fd14bd7/contracts/Staking.sol#L20

PLAYNITY

audit / code review report

December 15, 2021

TEAM RESPONSE

#1 - SafeMath not obligatory in Solidity v0.8+

OpenZeppelin SafeMath library is used for compatibility reasons. The version of the library being used is a component of OpenZeppelin library version 4.3.2, that is written for Solidity 8.x, and its implementation wraps the native Solidity math operations, so it has no negative effect on smart contracts security or performance.

#2 - Use reentrancy guard in more efficient way

In the scope of current use cases, the methods in question do not transfer native tokens (ETH/BNB) to arbitrary addresses nor call any unknown external contracts (the only external contracts being called are known and constant ERC20 token contracts), so there is inherently no possibility for reentrancy attacks and the reentrancy guards are actually redundant. Even if this was not the case, the outer methods logic (not directly protected by reentrancy guards) is designed to prevent reentrancy attacks (by placing the external contract calls after state change statements).

#3 - Check return values of erc20 transfer and transferFrom

In the scope of current use cases, the transfer and transferFrom methods are only called on known and constant ERC20 token contracts that properly implement those methods.

#4 - Additional check for feesTransferAddress

The feesTransferAddress can only be set by the contract owner and is not meant to be changed often. We are aware that care should be taken and additional verification performed when changing this parameter.

#5 - Emit events before external calls

In the scope of current use cases the current implementation will not cause any issues. Reentrancy is not a threat here as the methods in question do not transfer native tokens (ETH/BNB) to arbitrary addresses nor call any unknown external contracts (the only external contracts being called are known and constant ERC20 token contracts). Also, no event driven logic is currently used or planned regarding the methods concerned.

#6 - Recommendations for gas optimisation

a) By design, unstakingFee is a state variable that can be modified with setUnstakingFee method.

The methods concerned could have been in fact declared as external, however it should not have a considerable impact on gas consumption.

auditmos.com

Secure your space

contact@auditmos.com