shieldify

Dark Mythos

Balinor

SECURITY REVIEW

Date: 20 December 2023

CONTENTS

1. About Shieldify

We are Shieldify Security – revolutionizing Web3 Security. Elevating standards with top-tier reports, a unique subscription-based model.

Book a security review and learn more about us at shieldify.org or @ShieldifySec

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Dark Mythos

Dark Mythos is a fantasy trading card game that introduces NFT cards with a unique storytelling feature. NFT holders can unlock exclusive stories crafted by fantasy author Marco Dülk, delving deep into the lore of Dark Mythos and offering insights into the characters and their worlds. The game merges the thrill of collecting rare NFT cards with professionally written narratives, creating a personalized and immersive experience for fans of fantasy literature and trading card games. The stories are handcrafted by a skilled author to ensure authenticity and high literary quality, making each one a unique work of art.

Balinor is the name of the subsequent collection from the Dark Mythos team. This Smart Contract is designed for the Dark Mythos collections on the ShimmerEVM Chain, allowing for various minting mechanisms including purchasable minting, free claiming, and manager minting. It also integrates royalty management in compliance with the ERC721Royalty standard.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors

- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted four days and a total of 96 hours were spent on it by the core Shieldify team. This is the second audit for the protocol's subsequent NFT collection, which represents a single NFT contract of the ERC-721 standard. We would like to emphasize that this codebase is an improved version of the first one, as most of the security considerations from the first audit have been implemented by the developers. Considering the codebase's small size, the security review managed to identify issues, primarily around missing or improper input validation checks. The NatSpec is comprehensive.

We extend our gratitude to the Dark Mythos's team for their fast responsiveness, offering comprehensive clarifications and detailed fedback for our inquiries.

We would also like to point out that the project's network of choice – IOTA's Shimmer, is still a relatively unexplored territory in terms of performance and network-level bugs and issues that might create additional attack surfaces.

5.1 Protocol Summary

Project Name	**Dark Mythos**
Repository	Dark-Mythos
Type of Project	ERC-721
Audit Timeline	4 days
Review Commit Hash	cb3abbc72a189d1dd6d81444ee8dd54326cff0fa
Fixes Review Commit Hash	6218b31025026202196bf0d3a6dcac598a1f6446

5.2 Scope

The following smart contracts were in the scope of the security review:

contracts/DarkMythosBalinor.sol	199
Total	**199**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **0**
- **Medium** issues: **2**
- **Low** issues: **2**

ID	Title	Severity
[M-01]	Insufficient Input Data Validation	Medium
[M-02]	Centralization Risks	Medium
[L-01]	Gas-Griefing Possibility In `withdraw()` Method	Low
[L-02]	The `mint()` Method does Not Follow the `Checks-Effects-Interactions` Pattern	Low

7. Findings

[M-01] Insufficient Input Data Validation

Severity

Medium Risk

Description

The following state variables are not validated in any regard or missing additional validation: `royaltyFraction`, `maxClaimableSupply`, `maxManagerMintableSupply` and `vendor` in the constructor at initializing phase.

This can lead to the disfunction of the `DarkMythosBalinor.sol` due to the fact that all of these variables are immutable so they can not be changed after deployment time.

Example scenarios are:

1. The `royaltyFraction` parameter is represented as `10 000 (100%)`. The check asserts that the variable should be less than `10 000 (100%)` which could potentially allow the deployer to set the royalty fee to `100%` for secondary sales, which could impact the economics of the protocol and disincentive the community to trade the tokens. This on the other hand could lead to decreased trading volume and reduced longevity of the project.
2. If the deployer accidentally sets `_maxManagerMintableSupply` or `_maxClaimableSupply` to zero, `updateWhitelistClaimAllocations()` and `mint()` function will revert. Note: For `mint()` function, it will revert when `isManagerMint` or `isClaimMint` parameter is true.
3. The constructor checks if `_royaltyReceiver` and `_vendor` addresses are not `address(0)`. However, there is a missing check if these addresses are identical. After discussion with the Dark Mythos team, it was deduced that if the `_royaltyReceiver` and `_vendor` addresses are the same, this could lead to unwanted issues in the business logic and incorrect privileges for these actors.

Location of Affected Code

File: contracts/DarkMythosBalinor.sol#L93-L130

```
constructor(
  string memory _name,
  string memory _symbol,
  string memory _baseURI_,
  uint256 _mintingCost,
  uint256 _maxManagerMintableSupply,
  uint256 _maxClaimableSupply,
  uint256 _maxPurchasableSupply,
  uint256 _maxMintsPerPurchaseTx,
  uint256 _allowPurchaseAfter,
  address _vendor,
  address _royaltyReceiver,
  uint96 _royaltyFraction
) payable ERC721(_name, _symbol) Ownable(msg.sender) {
  .
  .
  require(_royaltyFraction <= 10000, "Royalty fraction must be <= 10000")
     ;
  .
  .
  require(_vendor != address(0), "Vendor must not be the zero address");
  require(_royaltyReceiver != address(0), "Royalty receiver must not be
     the zero address");
  .
  .
}
```

Recommendation

1. We propose that the max allowed amount for `_royaltyFraction` should be something not higher than **20%** `(2000)` to prevent such a case. This could be implemented with a small change in the check in the constructor:

```
+ uint256 private constant _MAX_ROYALTY_FRACTION_FEE = 2000; // 20%;

+ require(_royaltyFraction <= 2000, "Royalty fraction must be <= 20%");
```

2. It is recommended to add a check for `maxManagerMintableSupply != 0` and `maxClaimableSupply != 0` in the `constructor`.
3. It is recommended to add a check for `_royaltyReceiver != _vendor` in the `constructor` and `setRoyaltyReceiver()` functions.

Team Response

Acknowledged and fixed.

[M-02] Centralization Risks

Severity

Medium Risk

Description

In the current implementation setup, the owner is meant to be a single Externally Owned Account (EOA) which could potentially introduce security risks regarding single points of failure or mismanagement.

One possible attack scenario is if the owner is compromised a DoS attack on the protocol can be executed based on pausing functionality.

Additionally the following functions: `updateWhitelistClaimAllocations()`, `provideSupply()`, `setRoyaltyReceiver()`, `addManagerContract()`, `removeManagerContract()` are dependant solely on the owner and a single mistake can lead to serious impact.

Location of Affected Code

File: contracts/DarkMythosBalinor.sol

```
function updateWhitelistClaimAllocations(address[] calldata users,
   uint256[] calldata amounts) external onlyOwner {
function provideSupply(uint256 amount) external onlyOwner {
function setRoyaltyReceiver(address _royaltyReceiver) external payable
   onlyOwner {
function addManagerContract(address _manager) external payable onlyOwner
   {
function removeManagerContract(address _manager) external payable
   onlyOwner {
function pause() external payable onlyOwner {
function unpause() external payable onlyOwner {
```

Recommendation

It's crucial for the long-term success and trustworthiness of the project to address these concerns thoroughly. One possible solution is using a multi-sig or governance as the protocol owner. Additionally consider using a Timelock smart contract so that users know in advance the applied changes.

Team Response

Acknowledged.

[L-01] Gas-Griefing Possibility In `withdraw()` Method

Severity

Low Risk

Description

In the current implementation of the call in `withdraw()` function the data parameter is omitted, but the returned data from the receiver will be copied to memory by the contract at compile time anyway.

A malicious actor can execute a gas-griefing attack, but there is no economic incentive for the attacker since the money transfer to his address will fail.

However, we recommended preventing such a possibility since the `vendor` address might be a multi-signature wallet or relayer might be used, where internal sub-calls are executed. If the gas is insufficient the funds might become forever stuck in the contract because the `vendor` address cannot be changed.

Location of Affected Code

File: contracts/DarkMythosBalinor.sol#L241

```
function withdraw() external {
  require(msg.sender == vendor, "Caller must be vendor");
  uint256 amount = address(this).balance;

  emit Withdraw(amount);

  (bool success, ) = msg.sender.call{ value: amount }("");
  require(success, "Withdrawal failed");
}
```

Recommendation

Consider using a low-level assembly call since it does not automatically copy return data to memory.

```
function withdraw() external {
  require(msg.sender == vendor, "Caller must be vendor");
  uint256 amount = address(this).balance;
  emit Withdraw(amount);

- (bool success, ) = msg.sender.call{ value: amount }("");

+ bool success;
+ assembly {
+   success := call(gas(), msg.sender, amount, 0, 0, 0, 0)
+ }

  require(success, "Withdrawal failed");
}
```

Team Response

Acknowledged and fixed.

[L-02] The `mint()` Method does Not Follow the `Checks-Effects-Interactions` Pattern

Severity

Low Risk

Description

The `mint()` function in `DarkMythosBalinor.sol` contract does not follow the `Checks-Effects-Interactions` (CEI) pattern. It is recommended to always first change the state before doing external calls.

There is no potential financial loss in the `mint()` function in `DarkMythosBalinor.sol` contract since there is a `nonReentrant` modifier.

Location of Affected Code

File: contracts/DarkMythosBalinor.sol#L165-L166

```
} else if (isClaimMint) {
  require(amount <= whitelistClaimAllocations[msg.sender], "Not enough
     allocation to claim");
  require(maxClaimableSupply != 0, "Claiming not enabled");
  _mint(claimableTokenIds, msg.sender, amount);

  whitelistClaimAllocations[msg.sender] -= amount;
  totalAllocatedTokens -= amount;
}
```

Recommendation

Consider sticking to the `CEI(Checks-Effects-Interactions)` pattern in `mint()` function in the following way:

```
else if (isClaimMint) {
  require(amount <= whitelistClaimAllocations[msg.sender], "Not enough
     allocation to claim");
  require(maxClaimableSupply != 0, "Claiming not enabled");
+ whitelistClaimAllocations[msg.sender] -= amount;
+ totalAllocatedTokens -= amount;

  _mint(claimableTokenIds, msg.sender, amount);

- whitelistClaimAllocations[msg.sender] -= amount;
- totalAllocatedTokens -= amount;
}
```

Team Response

Acknowledged and fixed.

shieldify

Thank you!