TrickOrTreat Security Audit Report by MK (12th DEC 2024)

Medium Risk Findings

[M-1] Use of _mint() Function Leading to Irretrievable NFTs

Description:

The contract utilizes the _mint() function in several places to mint NFTs. However, this may lead to NFTs being sent to addresses that do not support ERC721 tokens, making them irretrievable. To mitigate this risk, _safeMint() should be used instead to ensure that the recipient address is capable of safely receiving the NFT.

Impact:

The use of the _mint() function may result in NFTs being sent to addresses that do not support ERC721 tokens. This can cause the NFTs to become permanently inaccessible, as they cannot be recovered or transferred from incompatible addresses, leading to a loss of assets.

Proof of Concept:

1. Line 81 and 110 does not implement `_safeMint` function.

```
81  _mint(address(this), tokenId);
110 _mint(recipient, tokenId);
```

Recommended Mitigation:

1. Replace the `_mint()` function with the `_safeMint()` function to ensure that the recipient address is capable of safely receiving the NFT.
2. Implement checks to confirm that the recipient address is ERC721-compliant before minting.
3. Consider adding additional logic to handle failed transfers, such as reverting the transaction if the minting fails or the recipient is incompatible.

Low Risk Findings

[L-1] Deprecation of transfer() for Ether Transfers

Description:

In \`withdrawFees()\`, the \`transfer()\` function is used to send Ether to the owner. This is not recommended, as \`transfer()\` only provides a fixed gas limit of 2300, is deprecated, and may become insufficient if EVM gas costs change in the future. This could result in the funds being stuck in the contract.

Impact:

The use of the deprecated `transfer()` function to send Ether may lead to failed transactions if gas limits become insufficient due to changes in EVM gas costs. This could cause the funds to be permanently stuck in the contract, preventing the owner from accessing them and leading to a loss of assets.

Proof of Concept:

1. transfer() has been deprecated and call() should be used instead.

```
function withdrawFees() public onlyOwner {
        uint256 balance = address(this).balance;
- Remove this  payable(owner()).transfer(balance);
+ Add this     (bool success,) = payable(owner()).call{value: balance}("");
+ Add this     require(success);
        emit FeeWithdrawn(owner(), balance);
    }
```

Recommended Mitigation:

Replace the deprecated transfer() function with the call() function to ensure that Ether transfers are handled correctly and can adapt to future changes in EVM gas costs. The call() function provides more flexibility, and the success of the transaction should be verified by checking the return value.

[L-2] Misleading `Swapped` Event Emitted During Minting Process

Description:

The functions trickOrTreat() and mintTreat() emit a misleading Swapped event despite no swap occurring. This could cause confusion in third-party applications, such as the frontend of the dApp, as the event incorrectly suggests a swap took place when only minting is happening.

Impact:

The emission of a misleading Swapped event during the minting process could confuse third-party applications, such as the frontend of the dApp. This may lead to incorrect user interface behaviors, as users or external systems may mistakenly believe a swap occurred, when in fact only an NFT was minted. This could compromise the user experience and the integrity of the application’s interactions with external platforms.

Proof of Concept:

1. The mistake can be found in `line 114`

```
114     emit Swapped(recipient, treat.name, tokenId);
```

Recommended Mitigations:

1. Update the `trickOrTreat()` and `mintTreat()` functions to emit a more accurate event, such as Minted, instead of the Swapped event. This will align the event with the actual functionality of the functions and prevent confusion in third-party applications.

For Example:

```
emit Minted(msg.sender, _treatName, tokenId);
```

By accurately reflecting the action being performed, the event will maintain clarity for users and external systems interacting with the dApp.

[L-3] Integer Division Vulnerability Allowing Free NFT Minting

Description:

The trickOrTreat function in the SpookySwap contract uses a multiplier to calculate minting costs. However, when the multiplier is 1/2 and treat.cost is less than 2 wei, integer division rounds the cost down to zero, allowing users to mint NFTs for free by sending any amount of ETH. This vulnerability bypasses the intended payment system, leading to potential financial loss for the contract owner.

Impact:

The integer division issue allows users to mint NFTs for free by bypassing the intended cost calculation, resulting in financial loss for the contract owner. This undermines the pricing structure and could lead to significant revenue loss if exploited.

Proof of Concept:

1. On `Line 71`

```
uint256 requiredCost = (treat.cost * costMultiplierNumerator) /
costMultiplierDenominator;
```

Recommended Mitigation:

To prevent requiredCost from being zero, introduce a sanity check after calculating the cost to ensure it is always greater than zero. This will avoid unintended zero-cost transactions and ensure that users always pay the intended amount for minting.

For Example:

```
uint256 requiredCost = (treat.cost * costMultiplierNumerator) /
costMultiplierDenominator;
require(requiredCost > 0, "Calculated cost is zero");
```

This ensures that the minting process always requires a valid payment, even in edge cases where the cost calculation could round down to zero.

[L-4] UnSafe NFT Minting and Transfer

Description:

The SpookySwap contract uses unsafe NFT transfer methods (`_mint` and `_transfer`) instead of their safe counterparts (`_safeMint` and `_safeTransfer`). This can lead to permanent loss of NFTs when they are transferred to contracts that don't support ERC721 token reception.

Impact:

The use of unsafe NFT transfer methods (_mint and _transfer) in the SpookySwap contract may result in NFTs being sent to contracts that do not support ERC721 tokens. This could cause the NFTs to become permanently irretrievable, leading to asset loss for users and diminishing trust in the platform.

Proof of Concept:

The contract implements NFT transfers in two critical functions:

1. In the `mintTreat` internal function:

```
function mintTreat(address recipient, Treat memory treat) internal {
    uint256 tokenId = nextTokenId;
    _mint(recipient, tokenId);  // Unsafe minting
    _setTokenURI(tokenId, treat.metadataURI);
    nextTokenId += 1;
}
```

2. In the `resolveTrick` function:

```
function resolveTrick(uint256 tokenId) public payable nonReentrant {
    // ... payment validation ...
    _transfer(address(this), msg.sender, tokenId);  // Unsafe transfer
    // ... cleanup code ...
}
```

Recommended Mitigation:

1. Replace _mint with _safeMint to ensure that NFTs are only minted to addresses that support ERC721 tokens.
2. Replace _transfer with _safeTransfer to verify that the recipient can safely receive the NFT during transfers.