Trick or Treat First Flight Audit Report

Lead Auditors:

- Aditya Mishra

Table of Contents

About the Project

SpookySwap is a Halloween-themed decentralized application where users can participate in a thrilling "Trick or Treat" experience! Swap ETH for special Halloween-themed NFT treats. But beware, you might get tricked! There's a small chance your treat will cost half the price, or you might have to pay double. Collect rare NFTs, trade them with friends, or hold onto them for spooky surprises. Will you be tricked or treated?

Actors

- **Owner/Admin (Trusted)** - Can add new treats, set treat costs, and withdraw collected fees.
- **User/Participant** - Can swap ETH for Halloween treat NFTs, experience "Trick or Treat", and trade NFTs with others.

Disclaimer

The Aditya Mishra team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

All Contracts in `src` are in scope.

```
src/
├── TrickOrTreat.sol
```

Compatibilities

- Blockchains: EVM Equivalent Chains Only
- Tokens: Native ETH

Known Issues

- In `_selectPresidentRecursive` there is an issue where if two candidates are tied, whoever was earlier in the list is dropped. This is known, and we are OK with it.
- There are other issues with this style of voting, like for example, in some cases a candidate who does worse will win. You can see a longer explainer here.
- We know that `1460 days` is not exactly 4 years. We are OK with that.

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	3
Low	6
Info	2
Total	13

High

[H-1] Weak Randomness

Description: The use of keccak256 hash functions on predictable values like block.timestamp, block.number, or similar data, including modulo operations on these values, should be avoided for generating randomness, as they are easily predictable and manipulable. The `PREVRANDAO` opcode also should not be used as a source of randomness. Instead, utilize Chainlink VRF for cryptographically secure and provably random values to ensure protocol integrity.

- Found in src/TrickOrTreat.sol Line: 57

```
            uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender, nextTokenId, block.prevrandao))) % 1000 + 1;
```

Impact:

1. Attackers can predict or manipulate the outcome of the randomness, leading to unfair advantages.
2. Vulnerable to front-running and other exploits, compromising the fairness and integrity of the protocol.

Proof of Concept:

- *Scenario:* An attacker could monitor the blockchain for transactions and manipulate the block timestamp or other parameters to influence the random number generation.

- *Exploit:* By controlling the block timestamp or mining a block, an attacker can ensure they receive a favorable outcome.

Recommended Mitigation:

1. Do not use `block.timestamp`, `now` or `blockhash` as a source of randomness
2. Try to use Chainlink VRF for cryptographically secure and provably random values to ensure protocol integrity.

[H-2] Functions send eth away from contract but performs no checks on any address.

Description: The contract sends ETH without verifying the recipient's address in two instances:

- trickOrTreat: Refunds excess ETH to `msg.sender`.

- withdrawFees: Transfers contract balance to the owner.

- Found in src/TrickOrTreat.sol Line: 48

```
    function trickOrTreat(string memory _treatName) public payable
nonReentrant {
```

- Found in src/TrickOrTreat.sol Line: 146

```
    function withdrawFees() public onlyOwner {
```

Impact:

1. If `msg.sender` is manipulated or if the owner is compromised, funds could be sent to unintended recipients.
2. Lack of address validation can lead to unauthorized withdrawals or refunds.

Proof of Concept:

- *Scenario:* An attacker could exploit a vulnerability in the contract or the Ethereum network to impersonate msg.sender or the owner.
- *Exploit:* By manipulating the contract state or transaction context, funds could be redirected to an attacker's address.

Recommended Mitigation:

- Implement checks to ensure that `msg.sender` is the intended recipient before sending ETH.
- Ensure the owner address is secure and regularly verified.
- Consider using `OpenZeppelin's Address.sendValue` for safer ETH transfers.

Medium

[M-1] Lack of input validation is found in functions `trickOrTreat` and `setTreatCost`.

Description: The contract does not adequately validate inputs, such as treat names in functions like `trickOrTreat` and `setTreatCost`. This can lead to unexpected behavior or errors.

- Found in src/TrickOrTreat.sol Line: 48

```
    function trickOrTreat(string memory _treatName) public payable
nonReentrant {
```

- Found in src/TrickOrTreat.sol Line: 34

```
    function setTreatCost(string memory _treatName, uint256 _cost) public
onlyOwner {
```

Impact:

- Invalid or malicious inputs can cause the contract to behave unpredictably or enter an unintended state.
- May allow attackers to exploit the contract by passing unexpected inputs, potentially leading to loss of funds or denial of service.

Proof of Concept:

- *Scenario:* A user calls trickOrTreat with a non-existent treat name.
- *Exploit:* The function could proceed with incorrect logic or revert unexpectedly, affecting user experience and contract functionality.

Recommended Mitigation:

- Implement checks to ensure inputs are valid and within expected ranges or formats.
- Use require statements to enforce input constraints and provide informative error messages.
- Conduct thorough testing with edge cases to ensure all inputs are handled correctly.

[M-2] Potential Reentrancy in `resolveTrick` Function

Description: The `resolveTrick` function transfers an NFT to the caller before cleaning up the storage related to pending NFTs. This sequence can potentially allow reentrancy attacks if the transferred NFT triggers a fallback function in a malicious contract.

- Found in src/TrickOrTreat.sol Line: 131

```
    _transfer(address(this), msg.sender, tokenId);
```

Impact: An attacker could exploit this by repeatedly calling `resolveTrick` through a fallback function, potentially leading to unauthorized access or manipulation of the contract state

Proof of Concept:

- *Scenario:* A malicious contract calls `resolveTrick` and uses a fallback function triggered by the NFT transfer to re-enter the function before storage cleanup.
- *Exploit:* The attacker could manipulate the contract state or drain resources by exploiting the reentrancy vulnerability.

Recommended Mitigation:

- Reorder the function logic to update the contract state before making any external calls.
- Ensure that the nonReentrant modifier is effectively used to prevent reentrant calls.

[M-3] Insufficient Event Emission for Critical Actions

Description: The contract lacks event emissions for some critical actions, such as when a treat's cost is updated or when a trick is resolved. This omission makes it difficult to track important state changes on the blockchain.

- Found in src/TrickOrTreat.sol Line: 45

  ```
          treatList[_treatName].cost = _cost;
  ```

Impact:

- Without events, users and developers cannot easily monitor or verify critical actions, reducing transparency and auditability.
- It becomes harder to debug and trace the contract's behavior without a comprehensive event log.

Proof of Concept:

- *Scenario:* A user updates a treat's cost using setTreatCost, but no event is emitted to log this change.
- *Exploit:* While not a direct exploit, the lack of events can lead to disputes or misunderstandings about the contract's state.

Recommended Mitigation:

- Emit events for all critical state changes, such as in `setTreatCost` and `resolveTrick`.
- Ensure events contain sufficient information to reconstruct the state change.
- Implement a consistent strategy for logging all significant actions and state changes.

Low

[L-1] Unsafe ERC20 Operations should not be used

ERC20 functions may not behave as expected. For example: return values are not always meaningful. It is recommended to use OpenZeppelin's SafeERC20 library.

- Found in src/TrickOrTreat.sol Line: 148

```
        payable(owner()).transfer(balance);
```

[L-2]: `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

- Found in src/TrickOrTreat.sol Line: 43

    ```
        function setTreatCost(string memory _treatName, uint256 _cost) public
    onlyOwner {
    ```

- Found in src/TrickOrTreat.sol Line: 48

    ```
        function trickOrTreat(string memory _treatName) public payable
    nonReentrant {
    ```

- Found in src/TrickOrTreat.sol Line: 118

    ```
        function resolveTrick(uint256 tokenId) public payable nonReentrant {
    ```

- Found in src/TrickOrTreat.sol Line: 146

    ```
        function withdrawFees() public onlyOwner {
    ```

- Found in src/TrickOrTreat.sol Line: 152

    ```
        function getTreats() public view returns (string[] memory) {
    ```

- Found in src/TrickOrTreat.sol Line: 156

    ```
        function changeOwner(address _newOwner) public onlyOwner {
    ```

[L-3]: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

- Found in src/TrickOrTreat.sol Line: 25

```
    event TreatAdded(string name, uint256 cost, string metadataURI);
```

- Found in src/TrickOrTreat.sol Line: 26

```
    event Swapped(address indexed user, string treatName, uint256 tokenId);
```

- Found in src/TrickOrTreat.sol Line: 27

```
    event FeeWithdrawn(address owner, uint256 amount);
```

[L-4]: Using `ERC721::_mint()` can be dangerous

Using `ERC721::_mint()` can mint ERC721 tokens to addresses which don't support ERC721 tokens. Use `_safeMint()` instead of `_mint()` for ERC721.

- Found in src/TrickOrTreat.sol Line: 81

```
            _mint(address(this), tokenId);
```

- Found in src/TrickOrTreat.sol Line: 110

```
        _mint(recipient, tokenId);
```

[L-5]: Costly operations inside loops.

Invoking `SSTORE`operations in loops may lead to Out-of-gas errors. Use a local variable to hold the loop computation result.

- Found in src/TrickOrTreat.sol Line: 32

```
        for (uint256 i = 0; i < treats.length; i++) {
```

[L-6]: State variable changes but no event is emitted.

State variable changes in this function but no event is emitted.

- Found in src/TrickOrTreat.sol Line: 43

```
    function setTreatCost(string memory _treatName, uint256 _cost) public
onlyOwner {
```

- Found in src/TrickOrTreat.sol Line: 118

```
    function resolveTrick(uint256 tokenId) public payable nonReentrant {
```

Informational

[I-1] Solidity pragma should be specific, not wide

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.0;`, use `pragma solidity 0.8.0;`

- Found in src/TrickOrTreat.sol Line: 2

```
pragma solidity ^0.8.24;
```

[I-2]: PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

- Found in src/TrickOrTreat.sol Line: 2

```
pragma solidity ^0.8.24;
```