our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Spellborne

SECURITY REVIEW

Date: 8 October 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and have secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

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 Spellborne

Monsters are core game entities that exist off-chain by default but can be minted as NFTs for trading or ownership tracking. This document explains how the backend and smart contracts interact to maintain monster state integrity.

Key Features

1. Royalties (ERC2981)
    - Enforced royalties on secondary sales
2. Token Locking (In-Game & Marketplace)
    - Locks prevent transfers (but allow mint/burn)
    - Used when a monster is in a squad or listed for sale
    - Only `MANAGER_ROLE` can lock/unlock
    - Prevents accidental transfers during gameplay
3. Standard NFT Functions:
    - `mint(address to)`: Mints a new NFT with an auto-incrementing tokenId
    - `burn(uint256 tokenId)`: Burns NFT (auto-unlocks if locked)
    - Both require `MINTER_ROLE`
4. Access Control:
    - `MINTER_ROLE`: Mint & burn tokens (backend wallet)
    - `MANAGER_ROLE`: Lock/unlock, royalties, pause, URIs
    - `DEFAULT_ADMIN_ROLE`: Grant/revoke roles

Lock Use Cases

- Squad: Locked when added to active squad, unlocked when removed
- Marketplace: Locked when listed for sale, unlocked after sale/expiry

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 2 days, with a total of 32 hours dedicated to the audit by the Shieldify team.

Overall, the code is well-written. The audit report contributed by identifying four Low severity issues, mainly related to failing to adhere to the EIP-7572 Standard, and missing `receive()` ETH functionality.

The Spellborne team has done a great job with their test suite and provided support and responses to all of the questions that the Shieldify researchers had.

5.1 Protocol Summary

Project Name	**Spellborne**
Repository	nft-contract
Type of Project	ERC721C NFT with Royalty Enforcement + Token Locking
Audit Timeline	2 days
Review Commit Hash	94226da94ef06ceeec906844e5b261b7934ee34a
Fixes Review Commit Hash	c0ed8eb85ded69c12128abc6a883e0e3941c478e

5.2 Scope

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

File	nSLOC
src/MonsterNFT.sol	130
Total	**130**

6. Findings Summary

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

- **Low** issues: `4`
- **Info** issues: `4`

ID	Title	Severity	Status
[L-01]	`ContractURIUpdated` Event Does Not Follow EIP-7572 Standard	Low	Fixed
[L-02]	Missing ETH `receive()` Function Despite Withdraw Functionality	Low	Fixed
[L-03]	Missing Event Emission in `setBaseURI()` Function	Low	Fixed
[L-04]	Centralization Risks	Low	Fixed

7. Findings

[L-01] `ContractURIUpdated` Event Does Not Follow EIP-7572 Standard

Severity

Low Risk

Description

The `ContractURIUpdated` event in the `MonsterNFT` contract does not follow the standard defined in eip-7572. The current implementation uses `event ContractURIUpdated(string uri);` while the EIP-7572 standard expects `event ContractURIUpdated();` without parameters. This non-standard event signature will cause marketplaces like OpenSea to miss contract URI update events, leading to outdated metadata display.

Location of Affected Code

File: src/MonsterNFT.sol#L50

```
event ContractURIUpdated(string uri);
```

Impact

The non-standard event signature causes marketplaces to miss contract URI update events, resulting in outdated metadata display, inconsistent information, reduced marketplace compatibility with OpenSea and other platforms, and a poor user experience where contract metadata updates are not reflected in marketplace listings.

Recommendation

Modify the event definition to follow the EIP-7572 standard:

```
event ContractURIUpdated();
```

Update the `setContractURI()` function to emit the event without parameters:

```
function setContractURI(string calldata uri) external onlyRole(
    MANAGER_ROLE) {
     _contractURIValue = uri;
     emit ContractURIUpdated();
}
```

Team Response

Fixed.

[L-02] Missing ETH `receive()` Function Despite Withdraw Functionality

Severity

Low Risk

Description

The `MonsterNFT` contract implements a `withdraw()` function that allows authorized users to withdraw ETH from the contract, indicating that the contract is designed to handle ETH. However, the contract lacks a `receive()`, `fallback()` or any other `payable` function, making it impossible for the contract to accept ETH through direct transfers.

This creates a functional gap where the contract has withdrawal functionality but no way to receive ETH.

Location of Affected Code

File: src/MonsterNFT.sol

```
// ============ WITHDRAW ============

/**
 * @notice Withdraw ETH from contract
 * @param amount Amount to withdraw
 */
function withdraw(uint256 amount) external onlyRole(MANAGER_ROLE)
   nonReentrant {
    require(address(this).balance >= amount, "Insufficient balance");
    (bool success, ) = payable(msg.sender).call{value: amount}("");
    require(success, "Transfer failed");
}
```

Impact

Attempts to send ETH to the contract will fail.

Proof of Concept

Add the following test to `test/MonsterNFT.t.sol`:

```
function test_cannot_receive_eth() public {
    vm.deal(address(manager), 10 ether);
    vm.prank(manager);
    (bool success, ) = payable(address(nft)).call{value: 1 ether}("");
    assertTrue(success, "Cannot receive ETH");
}
```

The other tests present in the file missed this issue as they directly use the `vm.deal` cheat code to assign ETH to the contract and then withdraw it.

Recommendation

Add a `receive()` function to allow the contract to accept ETH:

```
/// @notice Accept direct ETH transfers
receive() external payable {}
```

Alternatively, if ETH deposits are not intended, consider removing the `withdraw()` function or adding clear documentation explaining how ETH is expected to enter the contract.

Team Response

Fixed.

[L-03] Missing Event Emission in `setBaseURI()` Function

Severity

Low Risk

Description

The `setBaseURI()` function in the `MonsterNFT` contract updates the base URI for token metadata, but does not emit any event to notify marketplaces and indexers about the metadata update. This violates the https://eips.ethereum.org/EIPS/eip-4906 standard which expects a `BatchMetadataUpdate` event to be emitted when metadata is updated, as also expected by marketplaces like OpenSea.

Location of Affected Code

File: src/MonsterNFT.sol#L186-L188

```
function setBaseURI(string calldata uri) external onlyRole(MANAGER_ROLE)
   {
    _baseTokenURI = uri;
}
```

Impact

The missing event emission causes marketplaces and indexers to miss metadata updates, resulting in outdated token metadata being displayed to users, reduced marketplace compatibility, and poor user experience where base URI changes won't be reflected in token listings.

Recommendation

Add the `BatchMetadataUpdate` event emission following EIP-4906 standard:

```
event BatchMetadataUpdate(uint256 _fromTokenId, uint256 _toTokenId);

function setBaseURI(string calldata uri) external onlyRole(MANAGER_ROLE)
   {
    _baseTokenURI = uri;
    emit BatchMetadataUpdate(1, _nextTokenId - 1);
}
```

This ensures compatibility with marketplace expectations and proper metadata update detection for all existing tokens.

Team Response

Fixed.

[L-04] Centralization Risks

Severity

Low Risk

Description

The current design has centralization issues that put too much power behind a single key and make any key compromise more damaging.

1. The `MANAGER_ROLE` mixes powerful admin actions with simple automation (`lockToken()` , `unlockToken()`) as stated in the documentation:

> Backend calls contract to lock NFT (non-transferable).

> Backend unlocks NFT via contract call.

https://www.notion.so/monstudios/Monsters-283b08fdf0db80c28036c92bfa34abef

This means the backend key used for automation must also hold strong permissions (`pause()` / `unpause()` , `withdraw()` ETH, `setBaseURI()` , `setDefaultRoyalty()` and `setTokenRoyalty()`).

2. In addition, the constructor grants all roles to the deployer, so compromising the deployer gives full control.

Location of Affected Code

File: src/MonsterNFT.sol#L61-L78

```
// Constructor grants all roles to the deployer (single EOA)
constructor(
    string memory name_,
    string memory symbol_,
    string memory baseURI_,
    address royaltyRecipient_,
    uint96 feeNumerator_
)
    ERC721OpenZeppelin(name_, symbol_)
    BasicRoyalties(royaltyRecipient_, feeNumerator_)
{
    require(feeNumerator_ <= 10000, "Royalty fee exceeds 100%");
    _baseTokenURI = baseURI_;

    // Grant roles to deployer
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(MINTER_ROLE, msg.sender);
    _grantRole(MANAGER_ROLE, msg.sender);
}
```

File: src/MonsterNFT.sol#L126-L140

```
// Automation functions controlled by MANAGER_ROLE
function lockToken(uint256 tokenId) external onlyRole(MANAGER_ROLE) {
    require(_ownerOf(tokenId) != address(0), "Token does not exist");
    tokenLocked[tokenId] = true;
    emit TokenLockStatusChanged(tokenId, true);
}

function unlockToken(uint256 tokenId) external onlyRole(MANAGER_ROLE) {
    require(_ownerOf(tokenId) != address(0), "Token does not exist");
    tokenLocked[tokenId] = false;
    emit TokenLockStatusChanged(tokenId, false);
}
```

Impact

- If the backend key is compromised, an attacker can unpause, withdraw ETH, change royalties, change contract and token metadata, and lock/unlock tokens.
- If the deployer key is compromised, the attacker can control everything, including granting or revoking roles.

Recommendation

- Create an `OPERATOR_ROLE` only for `lockToken()` and `unlockToken()`. Keep admin actions under `MANAGER_ROLE`, ensure `DEFAULT_ADMIN_ROLE` and `MANAGER_ROLE` are moved to multisigs.
- Revoke the deployer's roles after deployment.

```
bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");

// Move automation functions to OPERATOR_ROLE
function lockToken(uint256 tokenId) external onlyRole(OPERATOR_ROLE) { /*
    ... */ }
function unlockToken(uint256 tokenId) external onlyRole(OPERATOR_ROLE) {
   /* ... */ }
```

Team Response

Fixed.

our shielding · Your smart contracts, our shielding · Your smart

shieldify

Thank you!