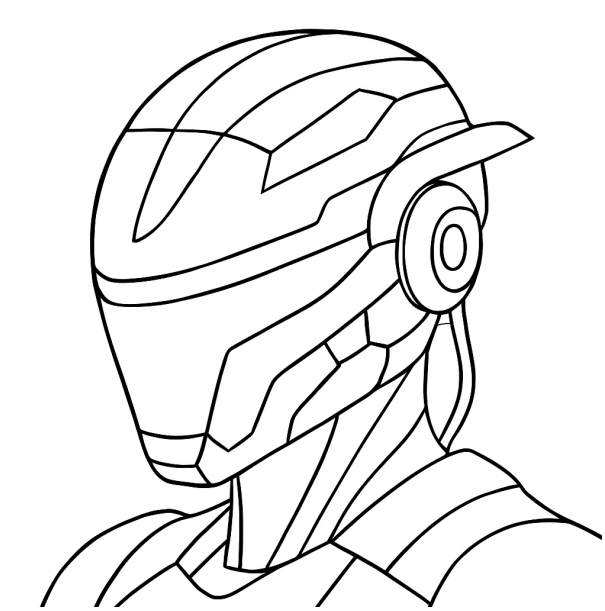

Beatland festival Audit Report

Version 1.0

X: @AlexScherbatyuk

July 24, 2025

Beatland festival Audit Report

Alexander Scherbatyuk

July 24, 2025

Prepared by: X: @AlexScherbatyuk Lead Auditors: - Alexander Scherbatyuk

Table of Contents

Protocol Summary

A festival NFT ecosystem on Ethereum where users purchase tiered passes (ERC1155), attend virtual(or not) performances to earn BEAT tokens (ERC20), and redeem unique memorabilia NFTs (integrated in the same ERC1155 contract) using BEAT tokens.

Disclaimer

The @AlexScherbatyuk 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
Likelihood	High	H	H/M	M
	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

```
src/
--- BeatToken.sol
--- FestivalPass.sol
--- interfaces
------ IFestivalPass.sol
```

Roles

Owner: The owner and deployer of contracts, sets the Organizer address, collects the festival proceeds.

Organizer: Configures performances and memorabilia.

Attendee: Customer that buys a pass and attends performances. They use rewards received for attending performances to buy memorabilia.

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	0
Info	1
Gas Optimizations	0
Total	2

Findings

Medium

[M-1] Memorabilia Collection: Last Item Cannot Be Redeemed Due to Incorrect Supply Bounds Check

Description The redeemMemorabilia function in FestivalPass.sol contains a logic error that prevents users from redeeming the last item in a memorabilia collection, even when supply is available.

Impact Users cannot redeem the final item in any memorabilia collection Collections effectively have 1 less item than their configured maxSupply This breaks the expected functionality where all items in a collection should be redeemable

Proof of Concepts * Create a memorabilia collection with maxSupply = 2 * Give a user sufficient BEAT tokens * Successfully redeem the first item * Attempt to redeem the second (and final) item * Expected: Success * Actual: Reverts with “Collection sold out”

```
// ============ Memorabilia Collection Supply Bounds Issue ============
    function test_MemorabiliaCollection_LastItemCannotBeRedeemed() public {
        // Create a collection with maxSupply = 2 (only 2 items available)
        vm.prank(organizer);
        uint256 collectionId = festivalPass.createMemorabiliaCollection(
            "Test Collection",
            "ipfs://test",
            100e18,
            2, // maxSupply = 2
            true
        );
        // Give user enough BEAT tokens
        vm.prank(address(festivalPass));
        beatToken.mint(user1, 300e18);
        // First redemption should succeed (item #1)
        vm.prank(user1);
        festivalPass.redeemMemorabilia(collectionId);
        // Verify first item was minted
        uint256 tokenId1 = festivalPass.encodeTokenId(collectionId, 1);
        assertEq(festivalPass.balanceOf(user1, tokenId1), 1, "First item should be minted");
        // Second redemption should succeed (item #2) - but it will fail due to the bug
        vm.prank(user1);
        vm.expectRevert("Collection sold out"); // This should NOT happen!
        festivalPass.redeemMemorabilia(collectionId);
        // The bug: when currentItemId = 2 and maxSupply = 2
        // The condition "currentItemId < maxSupply" becomes "2 < 2" which is false
        // But it should be "currentItemId <= maxSupply" which would be "2 <= 2" (true)
    }
```

Recommended mitigation Users will be able to redeem the final item in any memorabilia collection

```
- require(collection.currentItemId < collection.maxSupply, "Collection sold out");
+ require(collection.currentItemId <= collection.maxSupply, "Collection sold out");
```

Informational

[I-1] Immutable BeatToken Address Configuration

Description The FestivalPass constructor accepts a _beatToken address parameter but stores it as a public address variable without any setter function to modify it if incorrectly configured during deployment.

```
@> address public beatToken;
    address public organizer;
    .
    .
    .
@>    constructor(address _beatToken, address _organizer) ERC1155("ipfs
    ://beatdrop/{id}") Ownable(msg.sender) {
        setOrganizer(_organizer);
@>    beatToken = _beatToken;
    }
```

Impact In case the wrong _beatToken address is provided during deployment, the contract will be permanently linked to an incorrect or non-existent token contract. This would completely break core protocol functionality. The entire tokenomics and reward system would be non-functional. Users would lose access to earned rewards and bonuses. This represents a complete failure of the protocol's core value proposition.

Proof of Concepts This test demonstrates the core vulnerability by:

Deploying with Wrong Address: Creates a new FestivalPass contract with an incorrect beatToken address (0x1234567890123456789012345678901234567890)

Verifying the Problem: Confirms that the wrong address is permanently stored in the contract

Testing Core Functionality Failures:

- VIP Pass Purchase: When a user buys a VIP pass, the contract tries to mint 5 BEAT tokens to the user by calling BeatToken(beatToken).mint(msg.sender, bonus). Since beatToken points to a wrong address, this call fails.
- Performance Attendance: When users attend performances, the contract tries to mint reward tokens. This also fails due to the wrong address.
- Memorabilia Redemption: When users redeem memorabilia, the contract tries to burn BEAT tokens using BeatToken(beatToken).burnFrom(msg.sender, collection.priceInBeat). This fails as well.
- No Recovery Path: The test demonstrates that there's no way to fix this - no setter function exists to change the beatToken address.

```
    function test_Vulnerability_ImmutableBeatTokenAddress() public {
        // Deploy a new FestivalPass with an incorrect beatToken address
        address incorrectBeatToken = address(0x1234567890123456789012345678901234567890);
        FestivalPass vulnerableFestivalPass = new FestivalPass(incorrectBeatToken, organizer);
        // Verify the incorrect address is set
        assertEq(vulnerableFestivalPass.beatToken(), incorrectBeatToken);
        vm.prank(organizer);
        vulnerableFestivalPass.configurePass(2, VIP_PRICE, 1);
        // Try to buy a VIP pass - this will fail because the incorrect address doesn't have mint function
        vm.prank(user1);
        vm.expectRevert(); // Will revert when trying to call mint on incorrect address
        vulnerableFestivalPass.buyPass{value: VIP_PRICE}(2);
        // Try to attend a performance - this will also fail
        vm.prank(organizer);
        uint256 perfId = vulnerableFestivalPass.createPerformance(block.timestamp + 1 hours, 2 hours, 100e18);
        vm.warp(block.timestamp + 90 minutes);
        vm.prank(user1);
        vm.expectRevert(); // Will revert when trying to call mint on incorrect address
        vulnerableFestivalPass.attendPerformance(perfId);
        // Try to redeem memorabilia - this will also fail
        vm.prank(organizer);
        uint256 collectionId =
            vulnerableFestivalPass.createMemorabiliaCollection("Test Collection", "ipfs://test", 100e18, 10, true);
        vm.prank(user1);
        vm.expectRevert(); // Will revert when trying to call burnFrom on incorrect address
        vulnerableFestivalPass.redeemMemorabilia(collectionId);
        // There is NO way to fix this - no setter function exists
        // The contract is permanently broken if the wrong address is used during deployment
    }
```

Recommended mitigation Setter function to set a valid address after deployment with an onlyOwner modifier to restrict access rights to the owner only.

```
+     function setBeatToken(address _beatToken) external onlyOwner {
+         beatToken = _beatToken;
+     }
```