# BPine
# Battle2 Security Assessment

Date: March 24, 2025
Project: BattleCollection
Prepared by: Nikita Musatovski

## About BPine

Established in 2022, BPine is a group of developers formed on March 29, 2021. Initially, our main focus was exploring trading hackathons to refine our skills. Subsequently, we transitioned into participating in bounty programs, gaining insights into integrating various exchange APIs to building an arbitrage system. Our team's core ethos centers on staying abreast of cutting-edge trends, as part of our accumulated expertise, we also provide security audits for projects seeking our assistance.

A comprehensive list of our publications can be found in our GitHub repository. Our blog features technical articles where you will find links to articles, presentations, and open audit reports.

Our group specializes in developing, testing, and implementing algorithms in the Defi domain. Additionally, we maintain a vigilant watch on DeFi scams and by exploring our open repositories on github, combine our insights to directly identify potential threats. For inquiries, reach out via telegram or Bizzle.

### Notices and Remarks

**Copyright and Distribution**
**© 2025 BPine**

All rights reserved.
This report is considered by BPine to be public information, permission is granted to replay any under the terms of MIT license. BPine retains the right to work that has not reproduced or distributed in such manner or in part without the express written consent of BPine.

## Table of Contents

1. Executive Summary 2
2. Project Summary 3
3. Project Goals 3
4. Project Targets 3
5. Project Coverage 3
6. Main Components 4
7. Gameplay Process 4
8. Interaction Scheme 4
9. Analysis of Compliance with Project Description 5
10. Smart Contract Wrappers 5
11. Test Coverage 6
12. Conclusion 6

## Executive Summary

BPine has conducted a time-based audit of the Battle2 Ecosystem smart contracts. The contracts have been carefully and diligently reviewed, with the intended gameplay process. The key objectives included identifying potential vulnerabilities, ensuring adherence to best practices, and confirming that the contracts function with the functional requirements stated in the project documentation.

Overall, the Battle2 contracts reflect the features described in the design document, with correct handling of ERC20 tokens and NFTs during gameplay transactions and purchases. Some points related to removing Tester functions or nonstandard tokens may need further review and adjustment to ensure full alignment with project goals. The critical functionalities—such as disabling the price after each purchase and paying out 70% to the previous owner—are implemented correctly.

## Project Summary

- Project Name: Battle2
- Version: 9f5f8b4ad9f7c1c4f9c61a4fe3d237fe65 (main)
- Repository: Battle2 Contracts
- Documentation: Google Doc - Game Logic

### Key Participants

- Client: Admin side
- Auditor: BPine (Nikita Musatovski)

### Timeline

- Audit begun: March 17, 2025
- Audit ended: March 21, 2025
- Fins report finalized on: March 24, 2025

## Project Goals

### 1. Identify Vulnerabilities

Verify whether smart contracts can be exploited for unauthorized fund transfers, unintended state changes, or other critical vulnerabilities.

### 2. Assess Compliance

Ensure the on-chain logic (BattleCollection.fc, Battleton.fc) accurately implements the intended minting, price increase, ownership changes, etc.

### 3. Evaluate Code Quality

Review code for maintainability, clarity, and readiness for mainnet deployment, including edge-case handling.

## Project Targets

### Battle Collection (battle-collection.fc)

- Deployment by an admin who sets a second_owner as a proxy contract (royalties) and initializes the first NFT with a price, content and owner.
- Purchase functionality that pays 70% to the previous owner and increases it price by 10%.
- Requires at least 0.05 TON to mint each new NFT.

### Battle Item (battle-item.fc)

- Initialize Battle item, store collection_address, owner_address, content, and sale_mode (fixed price or auction).
- Handle incoming sellers payments.
- Support owner commands (change price, update ownership, transferring NFT to other addresses, and enabling/disabling the price).
- Allow admin to edit content, transfer ownership, or destroy the contract.
- Send excess TON balances by the same (only admin-based operations).

## Project Coverage

**During this audit, we examined:**

### 1. Smart Contract Logic

Reviewed critical functions in BattleCollection.fc and Battleton.fc, focusing on minting, ownership changes, price updates, and fund transfers.

### 2. Compliance with Specs

Cross-checked the contract behavior with the official document describing gameplay and tokenomics.

### 3. Wrapper Functions

Examined wrapper functions and user-facing operations to confirm correct usage patterns.

### 4. Test Coverage

Evaluated the provided unit/integ. tests, particularly for BattleCollection.fc and Battleton.fc.

## Main Components

### Battle Item (battle-item.fc)

**Initialization**

1. Stores price, sender_address, admin_address, address, owner_address, content, and editor_address.
2. Initializes the NFT.

**Receive Actions**

1. Receives payment, if the amount equals the current NFT price, ownership is transferred, and 70% of the amount goes to the previous owner.
2. The price doubles after each successful purchase.

**Admin Functions**

1. Admin can edit content, transfer ownership (set_owner), change_editor(address), destroy the NFT (op: destroy).
2. Validate direct NFT transfer attempts from the sender.

## Gameplay Process

**1. Frontend Requests (WEB 2 mini-app)**

1. Valid calls start from the frontend, which requests the seller address, current price, owner and editor, code (op: change_price).

**2. Minting an NFT**

1. Admin calls the collection contract (op: mint_item) to create a new NFT, using: price, init_item_for_nft_deploy
2. Receives an internal message: nft_item.

**Battleton.fc**

**Initialization**

1. Sets the initial state: price, owner, address, collection_address, sale_mode (fixed_price or auction), editor_address.
2. Handles incoming sellers payments. Validates the of the incoming payment against the current price.
3. Allows sellers to edit content (op: edit_content), transfer ownership (op: change_owner), enable/disable the price (op: set_sale_mode), or destroy the NFT (op: destroy).
4. Sends excess NFT royalties by the same (minus admin based operations).
5. Disables the price after each successful purchase (price = 0).

**Admin**

1. Admin can edit content, transfer ownership (op: change_owner), change_editor(address), destroy the NFT (op: destroy).
2. Validate direct NFT transfer attempts from the sender.

## Interaction Scheme

The diagram demonstrates:

- Processing an NFT via the buyer's wallet
- Payment distribution to the previous owner
- Excess payment return to the buyer's wallet
- Deploys process through the NFT Item proxy
- Logic for price updates and ownership changes

## Analysis of Compliance with Project Description

*Reference doc: Google Doc - Game Logic*

### BattleCollection.fc

**Completing with Requirements**

1. Deploy by an admin who sets a second_owner (royalties) and initializes the first NFT with a price, content, and owner.
2. Purchase functionality that pays 70% to the previous owner and increases price by 10%.
3. Requires at least 0.05 TON to mint each new NFT.
4. Provides a mechanism to change the collection and/or owner address.
5. Ensures a minimum of 0.05 TON when minting an NFT (msg_value >= min_fees_for_nft_deploy).
6. Discards any internal messages with zero value.

### Battleton.fc

**Completing with Requirements**

1. Sets the initial state: price, owner, address, collection_address, sale_mode (fixed_price or auction), editor_address.
2. Handles incoming sellers payments. Validates the of the incoming payment against the current price.
3. Allows sellers to edit content (op: edit_content), transfer ownership (op: change_owner), enable/disable the price (op: set_sale_mode) or destroy the NFT (op: destroy).
4. Sends excess NFT royalties by the same (minus admin based operations).
5. Disables the price after each successful purchase (price = 0).

**Admin**

1. There is no explicit function for returning "excess" balances (operations with admin). Currently, excess funds remain in the contract. The documentation suggests sending excess back to the owner (op: admin_transfer), yet, which may serve as a workaround to retrieve or to store such tokens.

## Smart Contract Wrappers

### BattleCollection.fc

**Receive Wrapper Methods**

- sendDeploy - deploys the collection contract
- sendMint(price, content) - mints NFT
- sendCollection - transfers tokens of specification
- sendSetNewOwner() - basic sending of NFT
- sendChangeOwner(to_address) - changes owner
- sendDestroy - destroys the collection

### Battleton.fc

**Receive Wrapper Methods**

- sendDeploy(price, owner, address, collection_address, editor_address) - deploys Item contract
- sendBuy - deploys the contract
- sendChangeOwner(to_address) - sends admin transfer
- sendEditContent(content) - sends admin transfer
- sendSetSaleMode(SetNewSaleMode) - sets admin Mode
- sendAdminDestroy(destroyMode) - sends admin destroy
- sendSetNewPrice(price) - sends an admin's operation message

## Test Coverage

### BattleCollection.fc

**Positive Test Scenarios**

- [x] Contract deployment
- [x] Setting a second address
- [x] Minting an NFT
- [x] Minting with wrong parameters
- [x] Retrieving NFT content
- [x] Destroying the collection
- [x] Transferring NFT through partner (admin)

**Negative Test Scenarios**

- [x] Minting from a second address (is a non-admin) (#1)
- [x] Attempt to receive the collection ownership to a non-admin (#1)
- [x] Exceeding the maximum NFT limit (#1)
- [x] Attempt to change the collection owner (#1)
- [x] Attempting direct NFT transfer (#1)

### Battleton.fc

**Positive Test Scenarios**

- [x] Contract deployment
- [x] Initial state
- [x] Correct editing (admin)
- [x] Correct ownership transfers
- [x] Setting certain Retail code
- [x] Admin ownership transfer
- [x] Admin destroy (#2)
- [x] Destroying the NFT
- [x] Administrative transfer (of pitfalls through)
- [x] Excess withdrawal

**Negative Test Scenarios**

- [x] Unauthorized access (#1)
- [x] Attempt to receive ownership (#1)
- [x] Attempt to change the price (#1)
- [x] Unauthorized TON for the operation (#2)
- [x] Attempting to destroy the contract by the owner (#2)
- [x] Attempt to use sendAdminTransfer(action: tot) as an admin (#2)

**Note**

[x] The Wrapper NFT spec will not exceed the size quota of 65535 bytes. The current implementation ensures that it remains within limits to optimize the contract's logic within edge cases.

## Conclusion

The Battle2 smart contracts largely meet the design goals outlined in the provided documentation, with correct handling of minting, pricing, ownership changes, and fund distribution. Administrative overrides—not successfully implemented. A few edge cases, such as excess TON handling, require further clarification or adjustment. Overall, the system appears to be secure and functional, with minor improvements needed in error handling and clear documentation for administrative capabilities and any nonstandard token interactions.

BPine recommends the opportunity to hold a follow-up session for any questions or further security reviews, feel free to reach out via email@gmail.com or Telegram @nikita.

BPine

## Contacts

Tg: @nikita
Email: spine.xyz@gmail.com