



SMART CONTRACT SECURITY AUDIT

ThunderNFT

Marketplace Contracts

CLIENT

ThunderNFT

PLATFORM

Fuel

LANGUAGE

Sway

REVIEW

October 2024

DOCUMENT

Document information

Project	ThunderNFT, Marketplace Contracts
Client	ThunderNFT
Platform	Fuel
Language	Sway
Repository	ThunderFuel/smart-contracts
Commit	260c9859e2cd28c188e8f6283469bcf57c9347de
Methods	Not recorded in the original report
Report date	October 2024
Fixes	No fixes implemented at the time of this audit

Document history

VERSION	DATE	CHANGE
1.0	October 2024	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	F-01 Front-Running Vulnerability Due to Protocol Fee Initialization Delay	8
	F-02 Funds Not Locked During Order Placement	9
	F-03 Cached Balance Vulnerability in Transfer Function	10
	F-04 Missing Logging in Ownership Functions	11
	F-05 Zero Balance Storage Retention in Withdrawals	12
	F-06 Ownership Assignment to Zero Address	12
	F-07 Royalty Fee Governance Flaw	13
07	Observations	14
08	Previous security audit reports	14
A	Appendix: vulnerability checklist	15
	Disclaimer	16

Executive summary

About ThunderNFT

The ThunderNFT protocol facilitates NFT auctions and sales by leveraging its unique ExecutionStrategy smart contract. It implements a fixed-price sale strategy and allows users to list, bid, and buy NFTs. The system also allows royalties management for NFT creators.

Summary of the review

Safe Edges reviewed ThunderNFT's Sway marketplace contracts on Fuel at commit 260c985, covering the execution strategy, exchange, pool, ownership and royalty management. Seven issues were identified. The original report did not assign severities, so none have been added here. The most significant concern a protocol fee that is unset at deployment, buy orders that do not lock funds, and a self-transfer that doubles a user's pool balance. No fixes had been implemented at the time of the report.

Findings by severity



Findings by status

STATUS	ISSUES
● Open	7

Scope

Repository	ThunderFuel/smart-contracts
Commit	260c9859e2cd28c188e8f6283469bcf57c9347de
Platform	Fuel
Language	Sway

IN-SCOPE FILES

```
ExecutionStrategy
ThunderExchange
Pool
Ownership Management
Royalty Manager
```

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Not recorded in the original report.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.
Not rated	The original report did not assign a severity to this finding. No rating has been added in this version.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
F-01	Front-Running Vulnerability Due to Protocol Fee Initialization Delay	Not rated	● Open
F-02	Funds Not Locked During Order Placement	Not rated	● Open
F-03	Cached Balance Vulnerability in Transfer Function	Not rated	● Open
F-04	Missing Logging in Ownership Functions	Not rated	● Open
F-05	Zero Balance Storage Retention in Withdrawals	Not rated	● Open
F-06	Ownership Assignment to Zero Address	Not rated	● Open
F-07	Royalty Fee Governance Flaw	Not rated	● Open

Detailed findings

F-01 Front-Running Vulnerability Due to Protocol Fee Initialization Delay

Severity Not rated Status ● Open

BRIEF

The contract is vulnerable to front-running attacks due to the protocol fee not being set during initialization. Attackers can exploit this delay between deployment and the protocol fee being set by submitting multiple fee-free orders.

IMPACT

Significant business logic impact to the protocol due to potential order flooding without fees.

PROOF OF CONCEPT

1. Deploy the contract with the protocol fee set to zero.
2. Submit multiple orders immediately after deployment but before the owner can call the `set_protocol_fee` function.
3. Attackers monitoring the mempool can rapidly submit orders without fees, exploiting this gap.

CODE REFERENCE

Reference: https://github.com/ThunderFuel/smart-contracts/blob/260c9859e2cd28c188e8f6283469bcf57c9347de/contracts-v1/execution_strategies/strategy_fixed_price_sale/src/main.sw#L54C1-L69C1

STRATEGY FIXED PRICE SALE

```
impl ExecutionStrategy for Contract {  
    /// Initializes the contract, sets the owner, and Thunder Exchange contract  
    #[storage(read, write)]  
    fn initialize(exchange: ContractId) {  
        require(!_is_initialized(), StrategyFixedPriceErrors::Initialized);  
        storage.is_initialized.write(true);  
  
        let caller = get_msg_sender_address_or_panic();  
        storage.owner.set_ownership(Identity::Address(caller));  
        storage.exchange.write(Option::Some(exchange));  
    }  
}
```

RECOMMENDATION

Modify the `initialize` function to require the protocol fee as a parameter and limit its value.

```
fn initialize(exchange: ContractId, fee: u64) {
    require(!_is_initialized(), StrategyFixedPriceErrors::Initialized);
    require(fee <= 500, StrategyFixedPriceErrors::FeeTooHigh);
    // Existing code to set ownership and exchange contract
}
```

F-02 Funds Not Locked During Order Placement

Severity Not rated Status ● Open

BRIEF

The `place_order` function in the ThunderExchange contract does not lock user funds during the order placement process. This flaw allows users to place multiple buy orders with the same balance, leading to potential losses.

IMPACT

Users can place multiple orders without sufficient funds, leading to financial losses for the platform.

PROOF OF CONCEPT

1. User A has 100 units in payment_asset.
2. They place a buy order for 100 units and quickly place another order before their balance is updated.
3. Both orders are accepted, leading to an overdraft.

CODE REFERENCE

Reference: https://github.com/ThunderFuel/smart-contracts/blob/260c9859e2cd28c188e8f6283469bcf57c9347de/contracts-v1/thunder_exchange/src/main.sw#L84C3-L132C6

PLACE ORDER FUNCTION, THUNDER EXCHANGE

```
fn place_order(order_input: MakerOrderInput) {
    _validate_maker_order_input(order_input);
    let strategy = abi(ExecutionStrategy, order_input.strategy.bits());
    let order = MakerOrder::new(order_input);
    match order.side {
        Side::Buy => {
            let pool_balance = _get_pool_balance(order.maker, order.payment_asset);
            require(order.price <= pool_balance,
ThunderExchangeErrors::AmountHigherThanPoolBalance);
        },
        Side::Sell => {
            // Validations
        },
    }
    strategy.place_order(order);
    log(OrderPlaced { order });
}
```

RECOMMENDATION

Introduce a mechanism to lock user funds before processing the order.

```
fn place_order(order_input: MakerOrderInput) {
    _validate_maker_order_input(order_input);
    let strategy = abi(ExecutionStrategy, order_input.strategy.bits());
    let order = MakerOrder::new(order_input);
    match order.side {
        Side::Buy => {
            let pool_balance = _get_pool_balance(order.maker, order.payment_asset);
            require(order.price <= pool_balance,
ThunderExchangeErrors::AmountHigherThanPoolBalance);
            _lock_funds(order.maker, order.payment_asset, order.price); // Lock the funds
        },
        // Side::Sell logic
    }
    strategy.place_order(order);
    log(OrderPlaced { order });
}
```

F-03 Cached Balance Vulnerability in Transfer Function

Severity Not rated Status ● Open

BRIEF

The `_transfer` function in the pool contract relies on cached balances for both the sender and recipient. This can lead to a double-spend or token duplication vulnerability.

IMPACT

Attackers can exploit cached balances, leading to token duplication and financial drain from the contract.

PROOF OF CONCEPT

When the sender and recipient addresses are identical, the cached balance is incremented twice, causing the user's balance to double.

CODE REFERENCE

Reference: <https://github.com/ThunderFuel/smart-contracts/blob/260c9859e2cd28c188e8f6283469bcf57c9347de/contracts-v1/pool/src/main.sw#L237C1-L257C2>

TRANSFER FUNCTION CODE

```
fn _transfer(from: Identity, to: Identity, asset: AssetId, amount: u64) {
    let from_balance = _balance_of(from, asset);
    let to_balance = _balance_of(to, asset);
    require(from_balance >= amount, PoolErrors::AmountHigherThanBalance);

    storage.balance_of.insert((from, asset), from_balance - amount);
    storage.balance_of.insert((to, asset), to_balance + amount);

    log(Transfer { from, to, asset, amount });
}
```

RECOMMENDATION

Check if the sender and recipient are the same and avoid balance update in that case.

```
fn _transfer(from: Identity, to: Identity, asset: AssetId, amount: u64) {
    if from == to {
        let balance = _balance_of(from, asset);
        require(balance >= amount, PoolErrors::AmountHigherThanBalance);
        log(Transfer { from, to, asset, amount });
        return;
    }
    // Proceed with regular transfer logic if from != to
}
```

F-04 Missing Logging in Ownership Functions

Severity Not rated Status ● Open

BRIEF

The `transfer_ownership` and `renounce_ownership` functions do not log events. This can lead to ownership changes being untracked, making it difficult to audit the contract.

IMPACT

Lack of auditability and security around ownership changes.

CODE REFERENCE

Reference: <https://github.com/ThunderFuel/smart-contracts/blob/260c9859e2cd28c188e8f6283469bcf57c9347de/contracts-v1/pool/src/main.sw#L197C3-L207C6>

TRANSFER OWNERSHIP FUNCTION

```
fn transfer_ownership(new_owner: Identity) {
    storage.owner.only_owner();
    storage.owner.transfer_ownership(new_owner);
}
```

RECOMMENDATION

Use the ownable library or manually add logging functionality for ownership transfers.

F-05 Zero Balance Storage Retention in Withdrawals

Severity Not rated Status ● Open

BRIEF

The withdraw function retains entries in storage for accounts with zero balances, leading to increased storage costs.

IMPACT

Unnecessary storage costs due to the retention of zero-balance entries.

RECOMMENDATION

Modify the withdraw function to remove zero-balance entries from storage.

```
if new_balance == 0 {
    storage.balance_of.remove(&(sender, asset));
} else {
    storage.balance_of.insert((sender, asset), new_balance);
}
```

F-06 Ownership Assignment to Zero Address

Severity Not rated Status ● Open

BRIEF

The contract allows ownership transfer to a zero address, which can lead to loss of control over the contract.

IMPACT

Loss of contract functionality due to an invalid owner.

RECOMMENDATION

Add a validation to prevent transferring ownership to the zero address.

```
require(!Ownership::is_zero_address(new_owner), AccessError::CannotReinitialized);
```

F-07 Royalty Fee Governance Flaw

Severity **Not rated** Status ● Open**BRIEF**

The contract allows the owner to set the royalty fee limit, which should be restricted to the Royalty Manager or NFT collection owner to ensure proper governance.

IMPACT

Misalignment between contract owners and NFT collection stakeholders.

RECOMMENDATION

Limit the ability to set royalty fees to the Royalty Manager or collection owner.

07

Observations

The ThunderNFT protocol's architecture allows for a flexible implementation, but care should be taken with regards to execution strategies and fund management. The lack of fund locking and delayed fee setup create vulnerabilities that must be addressed to ensure the integrity of the protocol.

This report provides a comprehensive review of the identified issues and recommended solutions for your smart contracts. Addressing these vulnerabilities is critical to ensuring the security and efficiency of the platform.

08

Previous security audit reports

No previous audits for ThunderNFT have been publicly disclosed.

APPENDIX A

Vulnerability checklist

Vulnerability classes checked during this review, as listed in the original report.

✓ Reentrancy	✓ External Dependency Risks
✓ Integer Overflow	✓ Timestamp Manipulation
✓ Integer Underflow	✓ Delegatecall to Untrusted Contracts
✓ Division by Zero	✓ Frozen Wallets
✓ Unauthorized Access Control	✓ Incorrect Implementation of Standards (e.g., ERC-20, ERC-721)
✓ Insufficient Gas Limit	✓ Insecure Oracle Usage
✓ Timestamp Dependency	✓ Lack of Multisig Authentication
✓ Transaction-Ordering Dependence (TOD)	✓ Incorrect Access Control Lists
✓ Front-Running	✓ Insecure Parameters
✓ Denial of Service (DoS)	✓ Lack of Upgradability Mechanism
✓ Logic Errors	✓ Improper Token Allowances
✓ Unhandled Exceptions	✓ Race Conditions
✓ Insecure Randomness	✓ Difficulty Adjustment Attacks
✓ Improper Error Handling	✓ Time Zone Vulnerabilities
✓ Lack of Input Validation	✓ Gas Limit Vulnerabilities
✓ Incomplete Delegation	✓ Lack of Proper Documentation
✓ Incorrect Balance Handling	✓ Misuse of Low-Level Calls
✓ Unintended Token Creation	✓ Dependency on External Contracts
✓ Incorrect Token Transfer	✓ Insider Attacks
✓ Misappropriation of Funds	✓ Phantom Token Attacks
✓ Function Visibility Misconfiguration	✓ Fallback Function Vulnerabilities
✓ Insecure External Calls	✓ Incorrect Constructor Logic
✓ Failure to Update State	✓ Chain Split Attacks
✓ Incorrect Event Handling	✓ Unprotected Private Data
✓ Unprotected Self-Destruct	✓ Insecure Compiler Version

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS