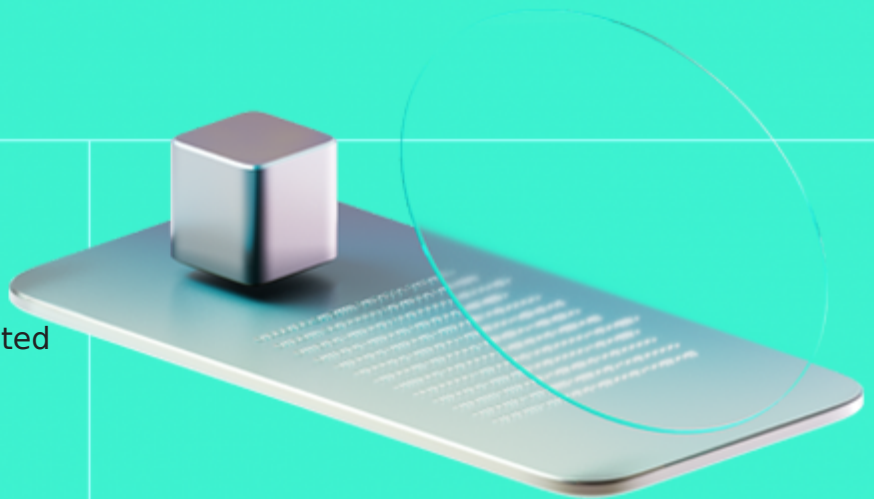




Smart Contract Code Review And Security Analysis Report

Customer: Aurora Labs Limited

Date: 27/01/2025



We express our gratitude to the Aurora Labs Limited team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

NEAR Intents is a protocol for multichain financial products built by Aurora.

Document

Name	Smart Contract Code Review and Security Analysis Report for Aurora Labs Limited
Audited By	Stepan Chekhovskoi
Approved By	Olesia Bilenka
Website	https://aurora.dev
Changelog	20/01/2025 - Initial Report
	27/01/2025 - Second Report
Platform	NEAR
Language	Rust
Tags	DeX, DeFi
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/near/intents
Initial Commit	160ba5829fea636283375ec462042d51409a6e66
Second Commit	91fee5e119fd74d8de1dbb57d27060873a0ae503

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

5	2	2	1
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	4
Low	1

Vulnerability	Severity
F-2025-8337 - Inability to Execute Swap due to Fee Calculation Mechanism	Medium
F-2025-8338 - Profitable intents Interception due to Front Running	Medium
F-2025-8343 - Lack of Full Access Key Verification for Fee Update	Medium
F-2025-8345 - Native Balance Exhausting due to Unpaid Storage Increase	Medium
F-2025-8339 - Lack of Signed Payload Versioning	Low

Documentation quality

- Functional overview is provided.
- Detailed functional requirements are missing.
- Technical description is partially provided.

Code quality

- Development environment is set up.
- Code lacks comments essential for complex architecture understanding.
- Code follows best coding practices.

Test coverage

Code coverage of the project is **22%** (region coverage).

- Core functionality and user interactions are tested.
- Negative cases coverage is partially missed.

Table of Contents

System Overview	6
Privileged Roles	6
Potential Risks	7
Findings	8
Vulnerability Details	8
Disclaimers	19
Appendix 1. Definitions	20
Severities	20
Potential Risks	20
Appendix 2. Scope	21
Appendix 3. Additional Valuables	25

System Overview

NEAR Intents allows users to create and manage their crypto portfolios, provides functionality of token swap directly connecting demand side with available liquidity provider. The protocol is deployed on NEAR blockchain.

NEAR Intents contract operates over signed intents what simplifies onboarding, allows users to delegate transaction execution and pay for Gas in various tokens. NEAR Intents contract supports ERC-191 (Ethereum wallets compatible), NEP-413 (NEAR wallets compatible), Raw Ed-25519 (Solana wallets compatible) signature types.

Onboarding Process

- Sending FT, NFT, or MT directly to the NEAR Intents contract causes appropriate callback is executed, the funds are wrapped into Multi Token Standard and are available for further actions.
- Additionally, signed intents can be attached to the deposit and are executed immediately. This allows to deposit funds, swap, and withdraw in the same transaction.

Swap Process

- The off-chain protocol part receives user request for token swap and passes it to Liquidity Providers to get a quote.
- The best quote is provided back to the user and the user signs the proposed amount.
- The signed intents are passed to the smart contract by the off-chain system part.
- The NEAR Intents contract validates the token balances delta over all intents is not negative and updates the actor balances.

It need to be noted here that the off-chain system part is not privileged user in the smart contract. The system is expected to integrate multiple request solvers and liquidity providers. Alternative off-chain modules can be run by community to avoid transactions censorship by the protocol administrators.

Admin functionalities

- The contract is upgradeable.
- The contract is pausable.
- The contract may take fee from each FT or MT swap.

Privileged roles

- The DAO is allowed to add or remove Full Access Key to the contract.
- The DAO and Fee Managers are allowed to set fee rate and fee collector contract.
- The DAO and Upgraders are allowed to upgrade the contract logic.
- The DAO and Unrestricted Withdrawers are allowed to withdraw any funds from the contract to arbitrary location.

Potential Risks

- **Continuous Increase of Contract Storage Deposit:** The contract allows any user to trigger storage allocation without paying for it. In terms of NEAR blockchain it is common to request Storage Deposit in NEAR (native coin) prior to allocating any storage for the user. However, the model brings difficulties in easy users onboarding, so, the development team acknowledged the risk of spreading storage by the users. The contract may reach DoS situation in case it has less funds than needed to cover the Contract Storage.
- **Centralization over DAO Account:** The DAO account is expected to be Multi-signature wallet controlling the system. The account is given wide permissions such as funds withdrawal to arbitrary destination and the smart contract upgrade.
- **Callback Failure causes Funds Lock:** The asynchronous nature of NEAR blockchain creates difficulties in error handling at third-party contracts execution. The system accepted possibility of funds being locked in case withdrawal was not successful due to abnormal Gas usage by the token contract. Considering simulating the transaction and attaching enough Gas to the withdrawal transaction.
- **JSON Redundant Fields:** The system parses signed messages payload as JSON. The payload may contain additional fields ignored by the smart contract which may potentially mislead the user regarding which action the message authorize.

Vulnerability Details

[F-2025-8337](#) - Inability to Execute Swap due to Fee Calculation Mechanism - Medium

Description:

The contract takes fee from each Token Diff instruction delta. Negative delta means user requests balance decrease and positive - balance increase. The fee is not implicitly included in the balance increase value and should be predicted off-chain based on fee rate and balance decrease value. After all the intents are executed, the contract verifies that sum of deltas for each token is zero.

```
/// core/src/intents/token_diff.rs

let amount = delta.unsigned_abs();
let fee = Self::token_fee(&token_id, amount, protocol_fee).fee_ceil(amount);

fees_collected
    .deposit(token_id, fee)
    .ok_or(DefuseError::BalanceOverflow)?;
```

This causes that fee is taken twice from each funds portion (first time from balance decrease and second time from balance increase). There might be impossible to find positive delta value for certain negative delta and fee rate.

This happens due to $\text{pos} + \text{ceil}(\text{pos} * \text{fee}) = -\text{neg} - \text{ceil}(-\text{neg} * \text{fee})$ equation is not always solvable relatively to integer `pos` for given integer `neg`.

This may lead to issues in resolving exact user swap requests. System might need to propose user to swap less tokens than was requested what might be unacceptable for tokens with little amount of decimals or small supply.

Example for `0.12%` fee.

- User negative delta is `-10026` of some token.
- Solver positive delta is `pos` of the same token.

At first the contract takes $\text{ceil}(10026 * 0.0012) = 13$ fee from negative delta and there is `10013` of token available.

- Then for `pos = 10000` plus fee `ceil(10000 * 0.0012) = 12` the needed amount is `10012` which is less than expected.
- And for `pos = 10001` plus fee `ceil(10001 * 0.0012) = 13` the needed amount is `10014` which is greater than expected.

So far, it is impossible to satisfy exact user request for given conditions. However, changing negative delta to `-10025` solves that particular case.

Assets:

- `core/src/engine/state/deltas.rs` [<https://github.com/near/intents>]
- `core/src/intents/token_diff.rs` [<https://github.com/near/intents>]

Status:

Fixed

Classification

Impact:	3/5
Likelihood:	4/5
Exploitability:	Independent
Complexity:	Medium
Severity:	Medium

Recommendations

Remediation: Consider weaken the “sum of deltas per token is zero” invariant to “sum of deltas per token is not positive” and send all excessive funds to fee collector.

The corresponding logic is located in the `finalize` function of the `core/src/engine/state/deltas.rs` file.

Alternatively take fees only from negative deltas.

```
// let amount = delta.unsigned_abs();
// let fee = Self::token_fee(&token_id, amount, protocol_fee).fee_ceil(amount);

// // collect fee
// fees_collected
//     .deposit(token_id, fee)
//     .ok_or(DefuseError::BalanceOverflow)?;

if delta < 0 {
    let fee = Self::token_fee(&token_id, delta.unsigned_abs(), protocol_fee)
```

```
        .fee_ceil(delta.unsigned_abs());  
    fees_collected  
    .deposit(token_id, fee)  
    .ok_or(DefuseError::BalanceOverflow)?;  
}
```

Resolution:

The Finding is fixed in the commit [877cb9562afa5d11a9b9f8213235be8f8e89695d](#).

The contract take fees only from negative delta token diffs (i.e. token_in).

[F-2025-8338](#) - Profitable intents Interception due to Front Running

- Medium

Description: According to the documentation, the off-chain request solvers might include intents containing self-reward to the transaction. The contract is free to process any combination of signed intents.

In case transaction contains a profitable intent, it can be intercepted by blockchain validator. The validator is able to create similar transaction changing the receiver of the profitable intent and put the malformed transaction in block before the initial one.

This may lead to any profitable intents be malformed by the validators and request solvers are not rewarded for the job done.

Assets:

- `core/src/intents/token_diff.rs` [<https://github.com/near/intents>]

Status: Mitigated

Classification

Impact: 3/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Simple

Severity: Medium


Recommendations

Remediation: Consider implementing a mechanism of intents are dependent on other intents.

Such mechanism requires updating the `TokenDiff` intent structure with new field containing list of intent hashes need to be executed in the same transaction.

Resolution: The Finding is mitigated according to the functionality update in the `e697962abcf0f4d6372981faea828e47b6e89078` commit.

Referral field is added to the `TokenDiff` intent. The referrers might be rewarded by some off-chain mechanism.



While this does not prevent front running possibility, the solution provides alternative way for solvers rewarding.

F-2025-8343 - Lack of Full Access Key Verification for Fee Update - Medium

Description:

The `set_fee` and `set_fee_collector` contract methods are restricted to the DAO and Fee Manager roles. However, the functions do not validate if the call is initialized by holder of Full Access or Function Call Key.

Function Call Keys are given to various applications to simplify user experience, however, the Function Call Key does not allow NEAR transfer from the account. Verifying if 1 yNEAR was attached guarantees the caller is authorized with Full Access Key.

```
fn set_fee(&mut self, #[allow(unused_mut)] mut fee: Pips) {
    require!(self.fees.fee != fee, "same");
    mem::swap(&mut self.fees.fee, &mut fee);
    FeeChangedEvent {
        old_fee: fee,
        new_fee: self.fees.fee,
    }
    .emit();
}

fn set_fee_collector(&mut self, #[allow(unused_mut)] mut fee_collector: AccountId) {
    require!(self.fees.fee_collector != fee_collector, "same");
    mem::swap(&mut self.fees.fee_collector, &mut fee_collector);
    FeeCollectorChangedEvent {
        old_fee_collector: fee_collector.into(),
        new_fee_collector: Cow::Borrowed(self.fees.fee_collector.as_ref()),
    }
    .emit();
}
```

This may lead to unauthorized application being able to set arbitrary fee in the contract and change the fee collector account.

Assets:

- defuse/src/contract/fees.rs [<https://github.com/near/intents>]

Status:

Fixed

Classification

Impact:

4/5

Likelihood: 4/5

Exploitability: Semi-Dependent

Complexity: Simple

Severity: Medium

Recommendations

Remediation: Consider validation the methods are executed with Full Access Key.

```
fn set_fee(&mut self, #[allow(unused_mut)] mut fee: Pips) {  
    assert_one_yocto();  
    ...  
}  
  
fn set_fee_collector(&mut self, #[allow(unused_mut)] mut fee_collector: AccountId) {  
    assert_one_yocto();  
    ...  
}
```

Resolution: The Finding is fixed in the commit [3eb11613f899dd375e2178d447e5c58a09e6a3b7](#).
The missing Full Access Key validation is implemented.

[F-2025-8345](#) - Native Balance Exhausting due to Unpaid Storage Increase - Medium

Description: NEAR blockchain implements storage staking model which locks some NEAR at contracts when additional storage space is allocated. It is considered reasonable to request storage deposits from users to cover the data increase costs.

The `execute_intents` function execution causes storage increase due to potential user balances allocation and used nonces recording. The function lacks validation if it was attached enough NEAR to cover the storage increase.

```
fn execute_intents(&mut self, signed: Vec<MultiPayload>) {  
    Engine::new(self, ExecuteInspector::default())  
        .execute_signed_intents(signed)  
        .unwrap_or_panic()  
        .as_mt_event()  
        .as_ref()  
        .map(MtEvent::emit);  
}
```

In such a way, third-party may perform a dust attack on the contract populating storage and decreasing the available amount of NEAR for covering the allocated space. This potentially may lead to temporary DoS situation.

Assets:

- defuse/src/contract/intents/mod.rs
[<https://github.com/near/intents>]

Status: Accepted

Classification

Impact: 3/5
Likelihood: 4/5
Exploitability: Independent
Complexity: Simple
Severity: Medium

Recommendations

Remediation:

Consider checking the initial and final contract state size in the `execute_intents` function of the `defuse/src/contract/intents/mod.rs` file. Ensure that enough NEAR were attached to the call to cover the state increase.

Resolution:

The Finding is accepted by the Client team.

The risk is applicable to the mentioned `execute_intents` and `on_[token]_transfer` onboarding functions.

To simplify the onboarding process, the `on_[token]_transfer` function does not require the initial storage deposit. The Unpaid Storage Increase issue cannot be fully fixed keeping users onboarding process easy.

[F-2025-8339](#) - Lack of Signed Payload Versioning - Low

Description: The signed payload messages are not properly versioned.

```
pub struct DefusePayload<T> {  
    pub signer_id: AccountId,  
    pub verifying_contract: AccountId,  
    pub deadline: Deadline,  
    #[serde_as(as = "Base64")]  
    #[cfg_attr(  
        all(feature = "abi", not(target_arch = "wasm32")),  
        schemars(example = "self::examples::nonce")  
    )]  
    pub nonce: Nonce,  
  
    #[serde(flatten)]  
    pub message: T,  
}
```

The system is developed as upgradeable. For upgradeable systems it is reasonable to implement versioning possibility from the beginning to simplify further support of different version payloads.

Assets:

- core/src/payload/mod.rs [<https://github.com/near/intents>]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 4/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Consider adding `version` field to the default payload structure and validate that it correspond to actual one in the `execute_signed_intent` function of the `core/src/engine/mod.rs` file.

```
pub struct DefusePayload<T> {  
    pub signer_id: AccountId,  
    pub verifying_contract: AccountId,  
    pub deadline: Deadline,  
    pub version: u64,  
    ...  
}
```

Resolution:

The Finding is accepted by the Client team.

Versioning system is going to be implemented in future updates.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

[hknio/severity-formula](https://github.com/hacken/severity-formula)

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project's performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/near/intents
Initial Commit	160ba5829fea636283375ec462042d51409a6e66
Second Commit	91fee5e119fd74d8de1dbb57d27060873a0ae503
Whitepaper	N/A
Requirements	https://docs.near-intents.org
Technical Requirements	README.md

Asset	Type
bitmap/src/lib.rs [https://github.com/near/intents]	Smart Contract
core/src/accounts.rs [https://github.com/near/intents]	Smart Contract
core/src/deadline.rs [https://github.com/near/intents]	Smart Contract
core/src/engine/inspector.rs [https://github.com/near/intents]	Smart Contract
core/src/engine/mod.rs [https://github.com/near/intents]	Smart Contract
core/src/engine/state/cached.rs [https://github.com/near/intents]	Smart Contract
core/src/engine/state/deltas.rs [https://github.com/near/intents]	Smart Contract
core/src/engine/state/mod.rs [https://github.com/near/intents]	Smart Contract
core/src/error.rs [https://github.com/near/intents]	Smart Contract
core/src/events.rs [https://github.com/near/intents]	Smart Contract
core/src/fees.rs [https://github.com/near/intents]	Smart Contract
core/src/intents/account.rs [https://github.com/near/intents]	Smart Contract
core/src/intents/mod.rs [https://github.com/near/intents]	Smart Contract
core/src/intents/token_diff.rs [https://github.com/near/intents]	Smart Contract

Asset	Type
core/src/intents/tokens.rs [https://github.com/near/intents]	Smart Contract
core/src/lib.rs [https://github.com/near/intents]	Smart Contract
core/src/nonce.rs [https://github.com/near/intents]	Smart Contract
core/src/payload/erc191.rs [https://github.com/near/intents]	Smart Contract
core/src/payload/mod.rs [https://github.com/near/intents]	Smart Contract
core/src/payload/multi.rs [https://github.com/near/intents]	Smart Contract
core/src/payload/nep413.rs [https://github.com/near/intents]	Smart Contract
core/src/payload/raw.rs [https://github.com/near/intents]	Smart Contract
core/src/tokens.rs [https://github.com/near/intents]	Smart Contract
crypto/src/curve/ed25519.rs [https://github.com/near/intents]	Smart Contract
crypto/src/curve/mod.rs [https://github.com/near/intents]	Smart Contract
crypto/src/curve/secp256k1.rs [https://github.com/near/intents]	Smart Contract
crypto/src/lib.rs [https://github.com/near/intents]	Smart Contract
crypto/src/payload.rs [https://github.com/near/intents]	Smart Contract
crypto/src/public_key.rs [https://github.com/near/intents]	Smart Contract
crypto/src/serde/curve.rs [https://github.com/near/intents]	Smart Contract
crypto/src/serde/mod.rs [https://github.com/near/intents]	Smart Contract
crypto/src/signature.rs [https://github.com/near/intents]	Smart Contract
defuse/src/accounts.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/abi.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/accounts/account.rs [https://github.com/near/intents]	Smart Contract

Asset	Type
defuse/src/contract/accounts/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/accounts/state.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/admin.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/config.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/events.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/fees.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/intents/execute.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/intents/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/intents/relay.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/intents/simulate.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/intents/state.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/state.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep141/deposit.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep141/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep141/native.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep141/withdraw.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep171/deposit.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep171/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep171/withdraw.rs [https://github.com/near/intents]	Smart Contract

Asset	Type
defuse/src/contract/tokens/nep245/core.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep245/deposit.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep245/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep245/resolver.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/tokens/nep245/withdraw.rs [https://github.com/near/intents]	Smart Contract
defuse/src/contract/upgrade.rs [https://github.com/near/intents]	Smart Contract
defuse/src/fees.rs [https://github.com/near/intents]	Smart Contract
defuse/src/intents.rs [https://github.com/near/intents]	Smart Contract
defuse/src/lib.rs [https://github.com/near/intents]	Smart Contract
defuse/src/tokens/mod.rs [https://github.com/near/intents]	Smart Contract
defuse/src/tokens/nep141.rs [https://github.com/near/intents]	Smart Contract
defuse/src/tokens/nep171.rs [https://github.com/near/intents]	Smart Contract
defuse/src/tokens/nep245.rs [https://github.com/near/intents]	Smart Contract
erc191/src/lib.rs [https://github.com/near/intents]	Smart Contract
nep413/src/lib.rs [https://github.com/near/intents]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.