

# Intellex

## Intellex Vault And Token Smart Contract Audit Report







# Document Control

**CONFIDENTIAL**

**REVIEW**<sub>(v1.1)</sub>

## Audit\_Report\_ITLX-VTK\_REVIEW\_11

Feb 17, 2025		v0.1	João Simões: Initial draft
Feb 18, 2025		v0.2	João Simões: Added findings
Feb 19, 2025		v1.0	Charles Dray: Approved
Apr 21, 2025		v1.1	João Simões: Reviewed findings

<b>Points of Contact</b>	Eric Hillerbrand	Intellex	erich@intellex.xyz
	Omar Saadoun	Intellex	omars@brainstems.ai
	Charles Dray	Resonance	charles@resonance.security
<b>Testing Team</b>	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Luis Arroyo	Resonance	luis.arroyo@resonance.security

## Copyright and Disclaimer

© 2025 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

# Contents

<b>1 Document Control</b>	<b>2</b>
Copyright and Disclaimer .....	2
<b>2 Executive Summary</b>	<b>4</b>
System Overview .....	4
Repository Coverage and Quality.....	4
<b>3 Target</b>	<b>6</b>
<b>4 Methodology</b>	<b>7</b>
Severity Rating.....	8
Repository Coverage and Quality Rating.....	9
<b>5 Findings</b>	<b>10</b>
Contract As Receiver Of ft_transfer().....	11
Unprotected Initialization Function Can Be Frontrun .....	12
Usage Of Outdated Packages .....	13
Usage Of Deprecated NEAR Collections .....	15
Redundant Contract State Validation.....	16
Missing Usage Of NEAR SDK Integer Types For Input And Output .....	17
Contract Pays One Yocto On Behalf Of Claimants .....	18
<b>A Proof of Concepts</b>	<b>19</b>

# Executive Summary

**Intellex** contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between February 12, 2023 and February 19, 2023. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 5 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Intellex with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.



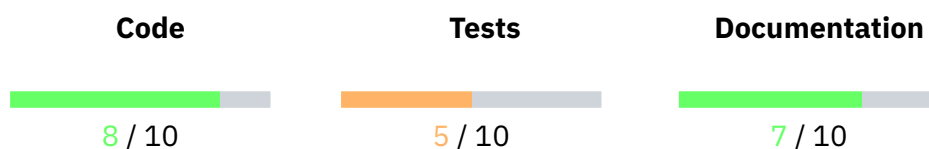
## System Overview

Intellex Protocol establishes a framework for efficiently managing interactions, allowing people to oversee, collaborate with, and securely engage with AI agents, while also enabling autonomous AI systems to coordinate, communicate, and work together to solve problems.

The Intellex NEAR smart contracts implement both a vesting mechanism through claimable vaults, as well as the associated fungible token to be used within the vault. Both of the implementations fork and follow industry standards, and possess no peculiar features.



## Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable but does not use the latest stable version of relevant components. Overall, **code quality is good**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is undetermined. Overall, **tests coverage and quality is average**.

- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good.**

# Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [brainstems/intellex\\_vesting\\_contracts/session\\_vault/src](#)
- Hash: fd83a4d76bfd0c4629d6d2409428db2921d76513
- Repository: [brainstems/itlx\\_nep141\\_token/src](#)
- Hash: 41778e9b991b7eb958e3ffa9f8fe7c9b4048b0b5

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

# Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

## Source Code Review - Rust NEAR

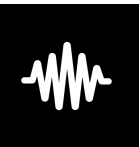
During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust NEAR audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Race conditions caused by asynchronous cross-contract calls
- Frontrunning attacks
- Storage staking
- Potentially problematic storage layout patterns
- Manual state rollbacks in callbacks
- Access control issues

- Denial of service
- Inaccurate business logic implementations
- Unoptimized Gas usage
- Arithmetic issues
- Client code interfacing



## Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info





# Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

# Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
- ■ **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ■■■ **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

---

RES-01	Contract As Receiver Of ft_transfer()	■■■ ■	Resolved
RES-02	Unprotected Initialization Function Can Be Frontrun	■■■■■	Resolved
RES-03	Usage Of Outdated Packages	■■ ■■■	Resolved
RES-04	Usage Of Deprecated NEAR Collections	■■■ ■	Resolved
RES-05	Redundant Contract State Validation	■■■■■	Resolved
RES-06	Missing Usage Of NEAR SDK Integer Types For Input And Output	■■■■■	Resolved
RES-07	Contract Pays One Yocto On Behalf Of Claimants	■■■■■	Resolved



# Contract As Receiver Of `ft_transfer()`

Medium RES-ITLX-VTK01

Business Logic

Resolved

## Code Section

- [session\\_vault/src/account.rs#L230-L255](#)
- [itlx\\_nep141\\_token/src/lib.rs#L99-L101](#)

## Description

The implemented NEP141 fungible token follows NEAR standards, and, as such, implements the function `ft_transfer()` used to transfer tokens between accounts.

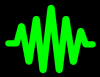
Due to the fact that this token is meant to be used within the `session_vault`, where tokens are meant to be transferred using `ft_transfer_call()`, it is important to ensure that users do not mistakenly call `ft_transfer()` to transfer tokens into the vault and miss on the accounting done on `ft_on_transfer()`, where the vault deposits are actually updated.

## Recommendation

It is recommended to implement a validation to ensure that during `ft_transfer()` function calls, the recipient cannot be the `session_vault` contract. Transfer to this vault should only be performed via `ft_transfer_call()`.

## Status

*The issue has been fixed in [118a96742cadd3da59176836ffab6a9a638b4d91](#) for the `intellex_vesting_contracts` repository and [27bd8a3caba87fd6dc3d15d02db3392b4728c042](#) for the `itlx_nep141_token` repository.*



# Unprotected Initialization Function Can Be Frontrun

Low

RES-ITLX-VTK02

Transaction Ordering

Resolved

## Code Section

- [session\\_vault/src/lib.rs#L55-L66](#)
- [itlx\\_nep141\\_token/src/lib.rs#L55-L70](#)
- [itlx\\_nep141\\_token/src/lib.rs#L74-L93](#)

## Description

The `new()` initialization function used to set up the protocol is unprotected. Anyone can frontrun and call this function to configure the protocol as they please. It is worth noting even after a successful exploitation the owner can still redeploy the contract to make adjustments as long as they have the key associated with the contract's `AccountId`.

Nevertheless, it is considered a best practice in the NEAR blockchain to mark the initialization functions with the macro `#[private]`. This enforces the predecessor `AccountId` to be equal to the contract itself, making it possible only for the contract's `AccountId` to call the function.

## Recommendation

It is recommended to mark the `new()` function as a private function.

## Status

*The issue has been fixed in [770b94af008faebaf2910275943605019c466bcb](#).*



# Usage Of Outdated Packages

Low

RES-ITLX-VTK03

Code Quality

Resolved

## Code Section

- Not specified.

## Description

The following Rust crates are used as dependencies of the project and contain known vulnerabilities:

- `chrono` (0.4.19). Potential segfault in `localtime_r` invocations. For more information: [RUSTSEC-2020-0159](#)
- `cranelift-codegen` (0.67.0, 0.68.0). Memory access due to code generation flaw in Cranelift module. For more information: [RUSTSEC-2021-0067](#)
- `curve25519-dalek` (3.2.1). Double Public Key Signing Function Oracle Attack on `ed25519-dalek`. For more information: [RUSTSEC-2024-0344](#)
- `ed25519-dalek` (1.0.1). Timing variability in `curve25519-dalek`'s `Scalar29::sub/Scalar52::sub`. For more information: [RUSTSEC-2022-0093](#)
- `idna` (0.2.3). `idna` accepts Punycode labels that do not produce any non-ASCII when decoded. For more information: [RUSTSEC-2024-0421](#)
- `raw-cpuid` (7.0.4). Optional Deserialize implementations lacking validation. For more information: [RUSTSEC-2021-0089](#)
- `raw-cpuid` (7.0.4). Soundness issues in `raw-cpuid`. For more information: [RUSTSEC-2021-0013](#)
- `remove_dir_all` (0.5.3). Race Condition Enabling Link Following and Time-of-check Time-of-use (TOCTOU). For more information: [RUSTSEC-2023-0018](#)
- `rocksdb` (0.15.0). Out-of-bounds read when opening multiple column families with TTL. For more information: [RUSTSEC-2022-0046](#)
- `shlex` (1.1.0). Multiple issues involving quote API. For more information: [RUSTSEC-2024-0006](#)
- `time` (0.1.43). Potential segfault in the `time` crate. For more information: [RUSTSEC-2020-0071](#)
- `wasmtime` (0.20.0). Multiple Vulnerabilities in Wasmtime. For more information: [RUSTSEC-2021-0110](#)
- `wasmtime` (0.20.0). Bug in pooling instance allocator. For more information: [RUSTSEC-2022-0075](#)
- `wasmtime` (0.20.0). Bug in Wasmtime implementation of pooling instance allocator. For more information: [RUSTSEC-2022-0076](#)
- `instant` (0.1.12). `instant` is unmaintained. For more information: [RUSTSEC-2024-0384](#)

- `mach` (0.3.2). `mach` is unmaintained. For more information: [RUSTSEC-2020-0168](#)
- `memmap` (0.7.0). `memmap` is unmaintained. For more information: [RUSTSEC-2020-0077](#)
- `parity-wasm` (0.41.0). Crate `parity-wasm` deprecated by the author. For more information: [RUSTSEC-2022-0061](#)
- `proc-macro-error` (1.0.4). `proc-macro-error` is unmaintained. For more information: [RUSTSEC-2024-0370](#)
- `wee_alloc` (0.4.5). `wee_alloc` is unmaintained. For more information: [RUSTSEC-2022-0054](#)
- `borsh` (0.8.2). Parsing `borsh` messages with ZST which are not-copy/clone is unsound. For more information: [RUSTSEC-2023-0033](#)
- `lock_api` (0.3.4). Some `lock_api` lock guard objects can cause data races. For more information: [RUSTSEC-2020-0070](#)
- `memoffset` (0.5.6). `memoffset` allows reading uninitialized memory. For more information: [RUSTSEC-2023-0045](#)
- `ahash` (0.4.7). `ahash` was yanked.
- `cpufeatures` (0.2.2). `cpufeatures` was yanked.
- `crossbeam-channel` (0.5.4). `crossbeam-channel` was yanked.
- `ed25519` (1.4.1). `ed25519` was yanked.
- `futures-util` (0.3.21). `futures-util` was yanked.
- `near-vm-runner` (4.0.0-pre.1). `near-vm-runner` was yanked.
- `parity-secp256k1` (0.7.0). `parity-secp256k1` was yanked.

It should also be noted that the NEAR SDK version 3.1.0 is also outdated and should be bumped to more recent versions to include the latest logical, performance, and security fixes.

## Recommendation

It is recommended to use more recent version of the identified crates that solve the identified security vulnerabilities, or otherwise stop using the dependency altogether.

## Status

*The issue has been fixed in 118a96742cadd3da59176836ffab6a9a638b4d91 for the intellex\_vesting\_contracts repository and 27bd8a3caba87fd6dc3d15d02db3392b4728c042 for the itlx\_nep141\_token repository. No further packages can be updated due to the near-workspaces dependency.*



# Usage Of Deprecated NEAR Collections

Info

RES-ITLX-VTK04

Code Quality

Resolved

## Code Section

- [session\\_vault/src/lib.rs#L38](#)
- [itlx\\_nep141\\_token/src/lib.rs#L41](#)

## Description

The smart contract makes use of the following deprecated collections from the NEAR SDK at `near_sdk::collections`:

- `collections::UnorderedMap`
- `collections::LazyOption`

A newer and more optimized version exists at `near_sdk::store`.

It should be noted that this is only possible by also bumping the Near SDK version.

## Recommendation

It is recommended to deprecate the usage of `near_sdk::collections` in favor of `near_sdk::store` equivalent components.

## Status

*The issue has been fixed in [118a96742cadd3da59176836ffab6a9a638b4d91](#) for the [intellex\\_vesting\\_contracts](#) repository and [27bd8a3caba87fd6dc3d15d02db3392b4728c042](#) for the [itlx\\_nep141\\_token](#) repository.*



# Redundant Contract State Validation

Info

RES-ITLX-VTK05

Gas Optimization

Resolved

## Code Section

- [session\\_vault/src/lib.rs#L56](#)
- [itlx\\_nep141\\_token/src/lib.rs#L76](#)

## Description

The initialization function `new()` marked with the macro `#[init]` begins with implementing code logic to ensure that the contract is already initialized. However, this same code logic is already implemented within the NEAR SDK thanks to the use of the macro `#[init]`.

## Recommendation

It is recommended to remove redundant state verification code from the initialization function.

## Status

*The issue has been fixed in [770b94af008faebaf2910275943605019c466bcb](#) for the [intellex\\_vesting\\_contracts](#) repository and [27bd8a3caba87fd6dc3d15d02db3392b4728c042](#) for the [itlx\\_nep141\\_token](#) repository.*





# Missing Usage Of NEAR SDK Integer Types For Input And Output

Info

RES-ITLX-VTK06

Code Quality

Resolved

## Code Section

- [session\\_vault/src/views.rs#L101](#)

## Description

The function `list_accounts()` makes use of input parameters of type `u64`. This type is longer than 52 bits and, as such, is not serializable by JSON, therefore making it impossible to retrieve a properly decoded value when calling this function through an external integrated interface.

## Recommendation

It is recommended to make use of NEAR SDK capitalized types in favor of the `i64-i128/u64-u128` native types when dealing with input and output parameters:

- `u64` - `U64`
- `u128` - `U128`
- `i64` - `I64`
- `i128` - `I128`

## Status

*The issue has been fixed in [118a96742cadd3da59176836ffab6a9a638b4d91](#).*



# Contract Pays One Yocto On Behalf Of Claimants

Info

RES-ITLX-VTK07

Business Logic

Resolved

## Code Section

- [session\\_vault/src/account.rs#L181](#)

## Description

The function `claim()` is used to claim fungible tokens that have vested for a session of time. When unclaimed tokens are available, a user can call this function and the tokens will be transferred to their account via the function `ft_transfer()`. Depending on the implementation of the fungible token's contract, it may or may not require one `yoctoNEAR` to be supplied.

In the case of this smart contract, `ONE_YOCTO` is being supplied into `ft_transfer()`, however, the contract's account is paying for it, whereas, the user should pay for claiming tokens. If used extensively, the `claim()` function can be used to drain contract funds.

## Recommendation

It is recommended to identify the `claim()` function as `#[payable]` and require one `yoctoNEAR` from the user.

## Status

*The issue has been fixed in `770b94af008faebaf2910275943605019c466bcb`.*

# Proof of Concepts

*No Proof-of-Concept was deemed relevant to describe findings in this engagement.*