SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Naramunz
Date: July 5th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Naramunz.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Near FT token;
Platform	Near
Language	Rust
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://whitepaper.naramunz.com/token
Timeline	23.05.2022 - 30.06.2022
Changelog	23.05.2022 - Initial Review 10.06.2022 - Second Review 20.06.2022 - Third Review 05.07.2022 - Forth Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Naramunz to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/CRG-AB/NARZ.git
Commit:
cc31589255673d2da68a3c8003151e30eeabcbbf
Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
Technical Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
JS tests: No
Contracts: NARZ/ft/*

Second review scope

Repository:
https://github.com/CRG-AB/NARZ.git
Commit:
fa8c3438f067ae971bfb729994d624815cfedada
Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
Technical Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
JS tests: No
Contracts: NARZ/ft/*

Third review scope

Repository:
https://github.com/CRG-AB/NARZ.git
Commit:
f72303c458476c3a76291b6c6bd0c6e0c9350d51
Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
Technical Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
JS tests: No
Contracts: NARZ/ft/*

Forth review scope

Repository:
https://github.com/CRG-AB/NARZ.git
Commit:
cea4146abb1f6e8f8e438507183706f21f5144a2
Documentation: Yes

https://docs.near.org/docs/roles/integrator/fungible-tokens
Technical Documentation: Yes
https://docs.near.org/docs/roles/integrator/fungible-tokens
JS tests: No
Contracts: NARZ/ft/*

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided the required functional and technical documentation. FT token documentation is not complete. The total Documentation Quality score is **9** out of **10**.

Code quality

The total CodeQuality score is **9** out of **10**. Most FT token functionality is hidden behind macros.

Architecture quality

The architecture quality score is **8** out of **10**. All the logic is implemented in one file. Some functions code could be moved to separate files to improve code readability.

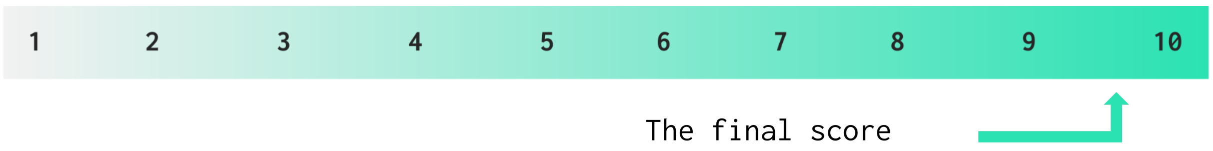
Security score

As a result of the fourth audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Description	Status
Missing Signer Checks	Case when instruction should only be available to a restricted set of entities, but the program does not verify that the call has been signed by the appropriate entity (e.g., by checking AccountInfo::is_signer).	Passed
Missing Ownership Checks	For accounts that are not supposed to be fully user-controlled, the program does not check the AccountInfo::owner field.	Passed
Redeployment with cross-instance confusion	The smart contract fails to ensure that the wasm code has the code it expects to have.	Passed
Arithmetic overflow/underflows	If an arithmetic operation results in a higher or lower value, the value will wrap around with two’s complement.	Passed
Numerical precision errors	Numeric calculations on floating point can cause precision errors, which can accumulate.	Passed
Loss of precision in calculation	Numeric calculations on integer types such as division can loss precision.	Passed
Casting truncation	Potential truncation problem with a cast conversion.	Passed
Exponential complexity in calculation	Finding computational complexity in calculations.	Passed
Over/under payment of loans	A loan overpayment is when paying extra towards a loan over and above the agreed monthly repayment. A loan underpayment is when paying less towards a loan over and below the agreed monthly repayment.	Passed
Anti-pattern function calls	Calling some anti-pattern function specific to Near blockchain.	Passed

Unsafe Rust code	The Rust type system does not check the memory safety of unsafe Rust code. Thus, if a smart contract contains any unsafe Rust code, it may still suffer from memory corruptions such as buffer overflows, use after frees, uninitialized memory, etc.	Passed
Outdated dependencies	Rust/Cargo makes it easy to manage dependencies, but the dependencies can be outdated or contain known security vulnerabilities. Cargo-outdated can be used to check outdated dependencies.	Passed
Redundant code	Repeated code or dead code that can be cleaned or simplified to reduce code complexity.	Passed
Do not follow security best practices	Failing to properly use assertions, check user errors, multisig, etc.	Passed
Project specification implementation check	Ensuring that the contract logic correctly implements the project specifications.	Passed
Contract-specific low-level vulnerabilities	Examining the code in detail for contract-specific low-level vulnerabilities.	Passed
Ruling out economic attacks	Economic rules that can be exploited to steal funds.	Passed
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Front-running or sandwiching	Checking for instructions that allow front-running or sandwiching attacks.	Passed
Unsafe design vulnerabilities	Checking for unsafe design which might lead to common vulnerabilities being introduced in the future.	Passed
As-of-yet Near unknown classes of vulnerabilities	Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Near blockchain.	Passed

Rug-pull mechanisms or hidden backdoors	Checking for rug-pull mechanisms or hidden backdoors.	Passed

System Overview

A standard interface for fungible tokens that allows for a normal transfer as well as a transfer and method call in a single transaction. The storage standard addresses the needs (and security) of storage staking.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

1. **Memory-exposure.**
   Memory access due to code generation flaw in cranelift module.
   A bug in 0.67.0 of the cranelift x64 backend can create a scenario that could result in a potential sandbox escape in a WebAssembly module. Cranelift-codegen version 0.67.0 is used in one of the contract dependencies.

   Contracts: NARZ/ft

   Recommendation: Consider Upgrading cranelift-codegen crate to >=0.73.1.

   Status: Fixed

2. **Memory-corruption.**
   In the affected version of this crate, the result of the race condition is that one or more tasks in the worker queue can be popped twice instead of other forgotten and never popped. Crossbeam-deque version 0.8.0 is used in one of the contract dependencies. Updating the crate could resolve the issue.

   Contracts: NARZ/ft

   Recommendation: Consider Upgrading crossbeam-deque crate to >=0.8.1.

   Status: Fixed

3. **Memory-corruption.**
   The arr! macro silently and unsoundly extends arbitrary lifetimes to 'static.
   fn unsound_lifetime_extension<'a, A>(a: &'a A) -> &'static A { arr![&A; a][0] }
   Generic-array version 0.12.3 is used in one of the contract dependencies.

   Contracts: NARZ/ft

   Recommendation: Consider Upgrading generic-array to >=0.8.4, <0.9.0 OR >=0.9.1, <0.10.0 OR >=0.10.1, <0.11.0 OR >=0.11.2, <0.12.0 OR >=0.12.4, <0.13.0 OR >=0.13.3

   Status: Fixed

4. **Denial-of-service**.
   The code contains several transmutes similar to this:
   https://github.com/gz/rust-cpuid/issues/40#issue-787745997
   Reading about transmutes in the nomicon, this is not correct because

the Rust compiler is free to reorder the struct fields.
Raw-cpuid version 7.0.4 is used in one of the contract dependencies.

Contracts: NARZ/ft

Recommendation: Consider Upgrading raw-cpuid crate to >=9.0.0.

Status: Fixed

5. **Denial-of-service**.
   Regexes with large repetitions on empty sub-expressions take a long time to parse.
   Regex version 1.4.3 is used in one of the contract dependencies.

   Contracts: NARZ/ft

   Recommendation: Consider Upgrading regex crate to >=1.5.5.

   Status: Fixed

6. **Memory-corruption**.
   Data race in Iter and IterMut. Read more about this issue
   thread_local version 1.1.3 is used in one of the contract dependencies.

   Contracts: NARZ/ft

   Recommendation: Consider Upgrading regex crate to >=1.1.4.

   Status: Fixed

7. **Memory-corruption.**
   There are multiple vulnerabilities in Wasmtime:
   -Use after free passing externrefs to Wasm in Wasmtime.
   -Out-of-bounds read/write and invalid free with externrefs and GC safepoints in Wasmtime.
   -Wrong type for Linker-define functions when used across two Engines.
   Dependency fungible-token-wrapper in near-sdk-sim uses Wasmtime version 0.20.0.

   Contracts: NARZ/ft

   Recommendation: Consider Upgrading wasmtime to >=0.30.0

   Status: Fixed

8. **Does not implement Drop.**
   Affected versions of this crate did not implement Drop when #[zeroize(drop)] was used on an enum. This can result in memory not being zeroed out after dropping it, which is exactly what is intended when adding this attribute.
   The flaw was corrected in version 1.2 and #[zeroize(drop)] on enums now properly implements Drop.
   Crate zeroize_derive version 1.0.1 is used instead of >=1.1.1.

   Contracts: NARZ/ft

   Recommendation: Consider upgrading zeroize_derive to >=1.1.1.

   Status: Fixed

9. **Potential segfault in the time crate.**
   There is a segfault detected in time crate version 0.1.44.
   Getenv and *setenv* in *libc* are not thread-safe, and most implementations of *localtime_r* in *libc* directly call *getenv*. This means that *localtime_r* may have data race with *setenv*.

   Contracts: NARZ/ft

   Recommendation: Consider upgrading the time crate to >=0.2.23

   Status: Fixed

■ Low

1. **Unused import**.
   Unused import `near_sdk::MockedBlockchain` in NARZ/ft/src/lib.rs.

   Contracts: NARZ/ft

   Recommendation: Remove unused imports to improve code readability.

   Status: Fixed

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.