Security Attestation and Secure Code Review

Findings and Recommendations Report Presented to:

NEAR Foundation

May 05, 2023
Version: 1.2

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

EXECUTIVE SUMMARY

Overview

NEAR Foundation engaged Kudelski Security to perform a secure code assessment and verification of technical claims implemented by the smart contract library known as "Near Contract Tools" being deployed to the NEAR blockchain.

In this report, we are making security attestations based on our review of code provided to us by the NEAR Foundation, written by team members from NEAR.

- This report contains no attestations, comments, or security review against the validity, functionality, or security of code in any commit hash other than:
  - **e08bf2db15280a1519c05ad9b57cdffc4e9e9fb2**

This assessment was conducted remotely by the Kudelski Security assigned resources and took place between November 2, 2022 and December 16, 2022. The result of our assessment is this report which:

- Provides the NEAR Foundation with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement;
- Provides a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place;
- Identifies potential issues and includes improvement recommendations if any functionality is not delivered to our satisfaction.

During the test, the following positive observations were noted regarding the scope of the engagement:

- The code was well-structured and well-commented
- Client contacts were highly collaborative with the Kudelski Security smart contract auditing team
- Tests were written for most functionality

This report summarizes the engagement and findings. This report also contains detailed descriptions of the proof of assertion, any vulnerabilities, steps the Kudelski Security teams took to identify each claim, as well as any applicable recommendations for remediation prior to final NEAR Contract Tools implementation.

Scope and Rules of Engagement

Kudelski Security performed a Security Attestation and Secure Code Review for NEAR Foundation. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

In-Scope Contracts for NEAR Contract Tools		
RBAC	**Approval**	**Events**
src/rbac.rs	src/approval/mod.rs	src/standard/nep297.rs
macros/src/rbac.rs	src/approval/native_transaction_action.rs	macros/src/standard/event.rs
	src/approval/simple_multisig.rs	macros/src/standard/nep297.rs
	macros/src/approval/mod.rs	macros/src/standard/mod.rs
	macros/src/approval/simple_multisig.rs	macros/src/rename.rs

Owner	Pause	Slot
src/owner.rs	src/pause.rs	src/slot.rs
macros/src/owner.rs	macros/src/pause.rs	

Upgrade	Fungible Token	Utils
src/upgrade/mod.rs	src/standard/mod.rs	src/utils.rs
src/upgrade/raw.rs	src/standard/nep141.rs	
src/upgrade/serialized.rs	src/standard/nep148.rs	
src/migrate.rs	macros/src/standard/fungible_token.rs	
macros/src/migrate.rs	macros/src/standard/nep141.rs	
	macros/src/standard/nep148.rs	

Table 1: NEAR Contract Tools Scope

NEAR Contract Tools Commit	
Github Commit Hash	e08bf2db15280a1519c05ad9b57cdffc4e9e9fb2

Table 1: NEAR Contract Tools Commit

Specific Methodology

Assessing a library is a bit different than directly auditing a smart contract, because the basic threat model is different. We must consider how the design of the library may increase or decrease the probability of a vulnerability occurring in the resulting smart contracts after multiple authors have used it. We can look at the past history of smart contract libraries and resulting exploits to provide some guidance in this regard.

Parties (diagram):

1. Creators: Library creators
2. Authors: Contract authors
3. Users: Contract users

The creator may have direct vulnerabilities such as unchecked math, or logical flaws. The authors may be more likely to inadvertently introduce vulnerabilities if the library is not designed with consideration to these vulnerabilities.

Attestations are generally not intended to determine if core functionality is implemented properly, but rather if security vulnerabilities, or the possibility thereof, have been considered. This is directly related to threat modeling. Thus, we do not guarantee that the code functions always as intended in the desired way, except as it pertains to the ability to exploit the code. The exception to this is when the core functionality is security functionality, such as in the RBAC component.

In general, we must check security-related invariants. These are assertions which we expect to be true but should be carefully checked to ensure that they hold under a variety of attacks, including fuzzing.

The form of the library takes several styles which must each be evaluated in their own way.

1. There are crates with structs that have been implemented in `impl` blocks, such as `slot.rs`. These follow best practices for library design.
2. There are traits which represent interfaces that are to be implemented (`rbac.rs`) and may have default implementations which can be evaluated. The authors may override the default implementation or add additional functions within the same scope that alter common data.
    - Attestations are based on and restricted to default usage of functions and data.
    - We will address non-default usage in the findings.
3. There is also example code such as `native_transaction_action` and `simple_multisig` which demonstrate usage of the library. This can be evaluated directly using best practices guides.

TECHNICAL ANALYSIS & FINDINGS

Components

The following table provides an overview of the core functionality of the library:

#	System	Description
1	RBAC	Contains functionality which allows assigning roles to accounts, checking if accounts have specific roles, and iterating over accounts corresponding to a specific role.
2	Events	Contains functionality which implements the NEP-297 event standard.
3	Approval	Contains functionality which allows implementation of multisig wallets including threshold approvals and native transaction actions
4	Owner	Contains functionality which controls storage and changing of contract ownership.
5	Pause	Contains functionality which enables reading and toggling a global pause state.
6	Slot	Contains functionality which involves creating and managing storage keys and reading and writing to a storage slot.
7	Upgrade	Contains functionality for upgrading contracts and migrating data to new schemas
8	Fungible Token	Contains functionality which implements the NEP-141 fungible token core standard and the NEP-148 fungible token metadata standard.
9	Utils	Contains utility functions for storage key generation and storage fee management.

Table 3: Functionality Overview

Attestations

The *Attestations* section provides an overview of each of the attestations, including explanation if applicable and relevant functions.

KSI-01 – RBAC

Role-based access control provides a critical feature of security, so the functionality in addition to possible vulnerabilities must be assessed.

#	Status	Description	Applicable Functions
1	Confirmed	Can only be called by the contract author	add_role, remove_role

We checked if #near_bindgen, #payable, or #ext_contract macros apply to any of the relevant code blocks

2	Confirmed	Successfully inserts role for account to collection in slot data	add_role
3	Confirmed	Given a role and account, role is properly removed from set in slot data	remove_role
4	Confirmed	Given an AccountId, the function always returns a correct boolean value or panics	has_role
5	Confirmed	Returns the slot corresponding to the set of all AccountIds assigned to a role	slot_members_of
6	Confirmed	Returns iterator over all members with specific role	iter_members_of
7	Confirmed	Returns the slot of the set of all the AccountIds assigned to a role after applying predicate to each member of the set	with_members_of
8	Confirmed	If AccountId does not exist in the set, the function returns false	has_role
9	Confirmed	Role with no members will just return an empty iterator	iter_members_of

Basic functionality is confirmed through tests.

10	Confirmed	If role already exists for account, it does not change slot data	add_role

11	Confirmed	If role does not exist for account, it does not change slot data	remove_role

These are security-related invariants. Changes to data beyond the specification may lead to vulnerabilities.

12	Confirmed	Panics when slot data is not properly serialized	add_role, remove_role
13	Confirmed	Panics if AccountId or role is not properly serialized in slot	has_role, require_role, prohibit_role
14	Confirmed	Panics when AccountId does not have specific role	require_role
15	Confirmed	Panics when AccountId has specific role	prohibit_role

Attackers may exploit code which does not panic. Proper panics prevent these kinds of attacks.

16	Confirmed	Invalid AccountIds are caught at compile time	has_role
17	Confirmed	All accountIds are validated (non-empty, etc.) at compile-time	All

Attackers may find vulnerabilities by providing invalid AccountIds. These are caught at compile time, preventing such attacks.

18	Confirmed	Invalid predicates and roles are caught at compile time	with_members_of

Attackers may find vulnerabilities by providing invalid predicates and/or rules. These are caught at compile time, preventing such attacks.

19	Confirmed	Does not change internal state	has_role, require_role, prohibit_role, slot_members_of

These functions should not change the slot data in any way.

Table 4: src/rbac.rs attestations and explanations

KSI-02 – Events

#	Status	Description	Applicable Functions
1	Confirmed	Prefixes "EVENT_JSON:"	to_event_string
2	Confirmed	Aborts if a string cannot be created	to_event_string
3	Confirmed	Logs the string, storing the message on chain	emit

Basic functionality is confirmed through tests.

Table 5: Events attestations and explanations

KSI-03 – Approval

For these attestations, we assume contract authors use the default implementations of these trait functions and do not access member storage slots using any other functions.

mod.rs

#	Status	Description	Applicable Functions
1	Confirmed	Returns a descriptive error if init is not called at runtime	get_config

Table 6: src/approval/mod.rs attestations and explanations

simple_multisig.rs

#	Status	Description	Applicable Functions
2	Confirmed	Checks whether an AccountId is allowed to submit an approval	is_account_authorized
3	Confirmed	Checks if valid signatures are greater than or equal to threshold	is_approved_for_execution
4	Confirmed	Returns error if there are repeat approvals	try_approve_with_authorized_account
5	Confirmed	Adds the account to the list of approvals	try_approve_with_authorized_account

Basic functionality is confirmed through tests

6	Confirmed	Checks that execution is occurring within the validity period	is_approved_for_execution, is_removable, try_approve_with_authorized_account
7	Confirmed	Does not change internal state	has_role, require_role, prohibit_role, slot_members_of

Table 7: src/approval/simple_multisig.rs attestations and explanations

native_transaction_action.rs

#	Status	Description	Applicable Functions
1	Confirmed	Implements eight NEAR actions for multisig wallets	execute

Basic functionality is confirmed through tests.

Table 8: src/approval/native_transaction_action.rs attestations and explanations

KSI-04 – Owner

For these attestations, we assume contract authors use the default implementations of these trait functions and do not access member storage slots using any other functions.

#	Status	Description	Applicable Functions
1	Confirmed	By default, only five functions are externally callable	own_get_owner, own_get_proposed_owner, own_renounce_owner, own_propose_owner, own_accept_owner

Table 9: Owner attestations

KSI-05 – Pause

For these attestations, we assume contract authors use the default implementations of these trait functions and do not access member storage slots using any other functions.

#	Status	Description	Applicable Functions
1	Confirmed	True only if paused, false otherwise	is_paused
2	Confirmed	Pauses if unpaused, otherwise panics	pause
3	Confirmed	Unpauses if paused, otherwise panics	unpause

Table 9: Pause attestations

KSI-06 – Slot

#	Status	Description	Applicable Functions
1	Confirmed	Writes raw bytes into the slot and aborts if unsuccessful	write_raw
2	Confirmed	Writes a value into the slot and panics if unsuccessful	write

Write functionality is confirmed through tests.

3	Confirmed	Reads raw data from the slot and aborts if unsuccessful	read_raw
4	Confirmed	Reads a value from the slot and panics if unsuccessful	read

Read functionality is confirmed through tests.

Table 10: Slot attestations

KSI-07 – Upgrade

For these attestations, we assume contract authors use the default implementations of these trait functions and do not access member storage slots using any other functions.

migrate.rs

#	Status	Description	Applicable Functions
1	Confirmed	The contract state must conform to the old schema otherwise deserializing it will panic	on_migrate
	Confirmed	Will panic on general serialization/deserialization issues	deserialize_old_schema

Table 11: src/migrate.rs attestations

raw.rs

#	Status	Description	Applicable Functions
1	Confirmed	Batches together to fail upgrade if migration fails	upgrade

Table 12: src/upgrade/raw.rs attestations

serialized.rs

#	Status	Description	Applicable Functions
1	Confirmed	Creates a promise that upgrades the current contract with the given code	upgrade

Table 13: src/upgrade/serialzed.rs attestations

KSI-08 – Fungible Token

For these attestations, we assume contract authors use the default implementations of these trait functions and do not access member storage slots using any other functions.

#	Status	Description	Applicable Functions
1	Confirmed	Panics on transfer of negative amount	transfer, transfer_unchecked, transfer_call
2	Confirmed	Panics if minimum deposit is not sent	deposit_unchecked, mint
3	Confirmed	Panics if transfer is called with invalid AccountId	transfer
4	Confirmed	Panics if sender has insufficient balance	withdraw_unchecked
5	Confirmed	Panics if receiver has too high a balance	deposit_unchecked

Table 14: Fungible Token (nep141.rs and nep148.rs) attestations and explanations

Findings

The Findings section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

During the Secure Code Review, we identified **2 Low** and **2 Informational** findings according to our Vulnerability Scoring System all of which were resolved.

The following chart provides an overview of the findings by severity:

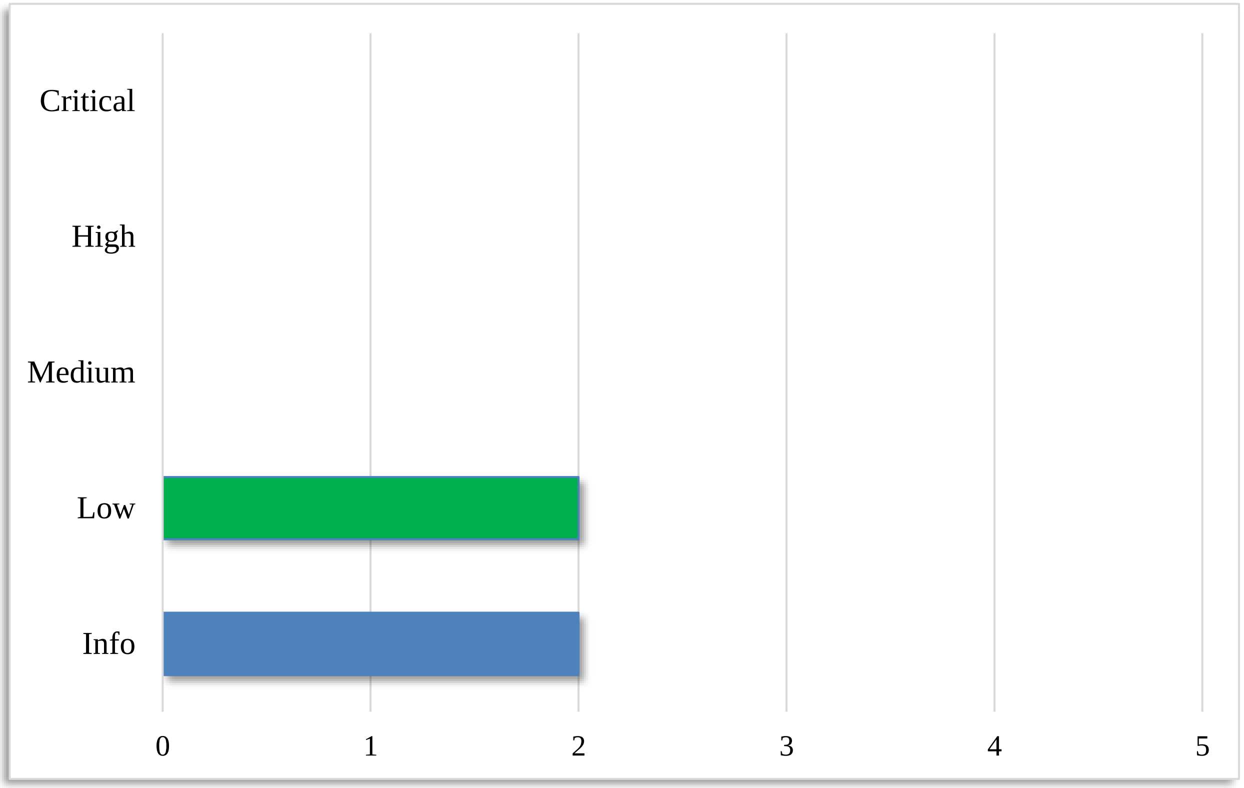


Figure 1: Findings by Severity

The following table provides an overview of the findings.

#	Severity	Description	Status
KS-09	**Low**	Event name collisions allowed	**Resolved**
KS-10	**Low**	Unchecked math	**Resolved**
KS-11	**Informational**	Renaming may remove extra characters	**Resolved**
KS-12	**Informational**	Missing test coverage	**Resolved**

Table 15: Findings Overview

KSI-09 – Event name collisions allowed

Severity	**LOW**
Status	**Resolved**

Impact	**Likelihood**	**Difficulty**
Low	Low	Medium

Description

Two different events may have identical names, either initially or through renaming.

Impact

If some infrastructure outside the blockchain is expecting a unique name, that infrastructure might misinterpret or entirely miss the event.

Evidence

Setting “name” in events or “rename” allows identical names for two different events. This can occur by explicitly setting identical names or renaming with an algorithm that has a non-unique mapping. For example, in `RenameStrategy::SnakeCase`, “Cat Dog”.to_snake_case() and “cat dog”.to_snake_case() both yield “cat_dog”.

Recommendation

Check for uniqueness, each time a new event name is assigned. This can be done by keeping a list of unique event names which can be validated against each time a new event is defined.

Remediation

This was resolved and merged in pull request 108: Disallows Event name collisions

KSI-10 – Unchecked math

Severity	**LOW**
Status	**Resolved**

Impact	**Likelihood**	**Difficulty**
Low	Low	Medium

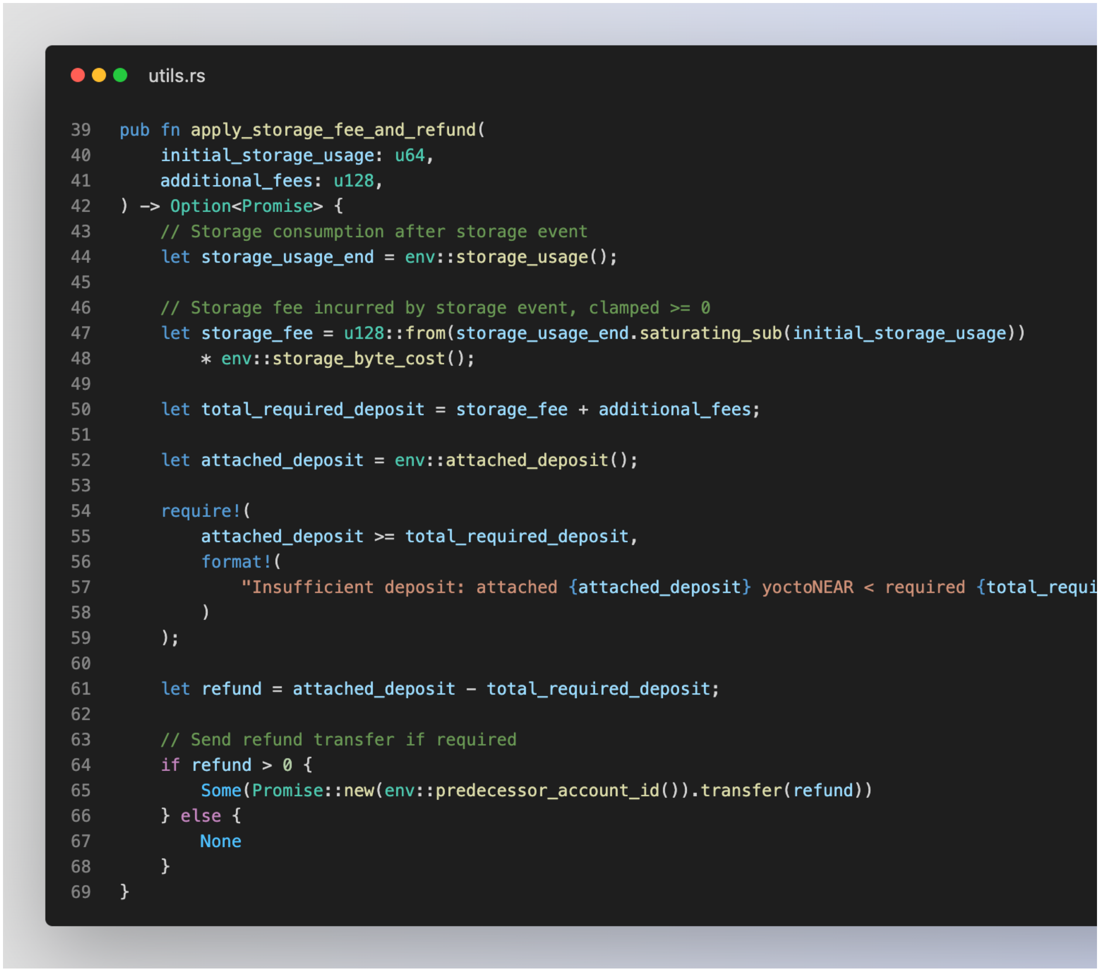
Description

Unchecked math was found in several files.

Impact

Overflows may lead to unpredictable results, such as higher than expected refund to predecessor.

Evidence



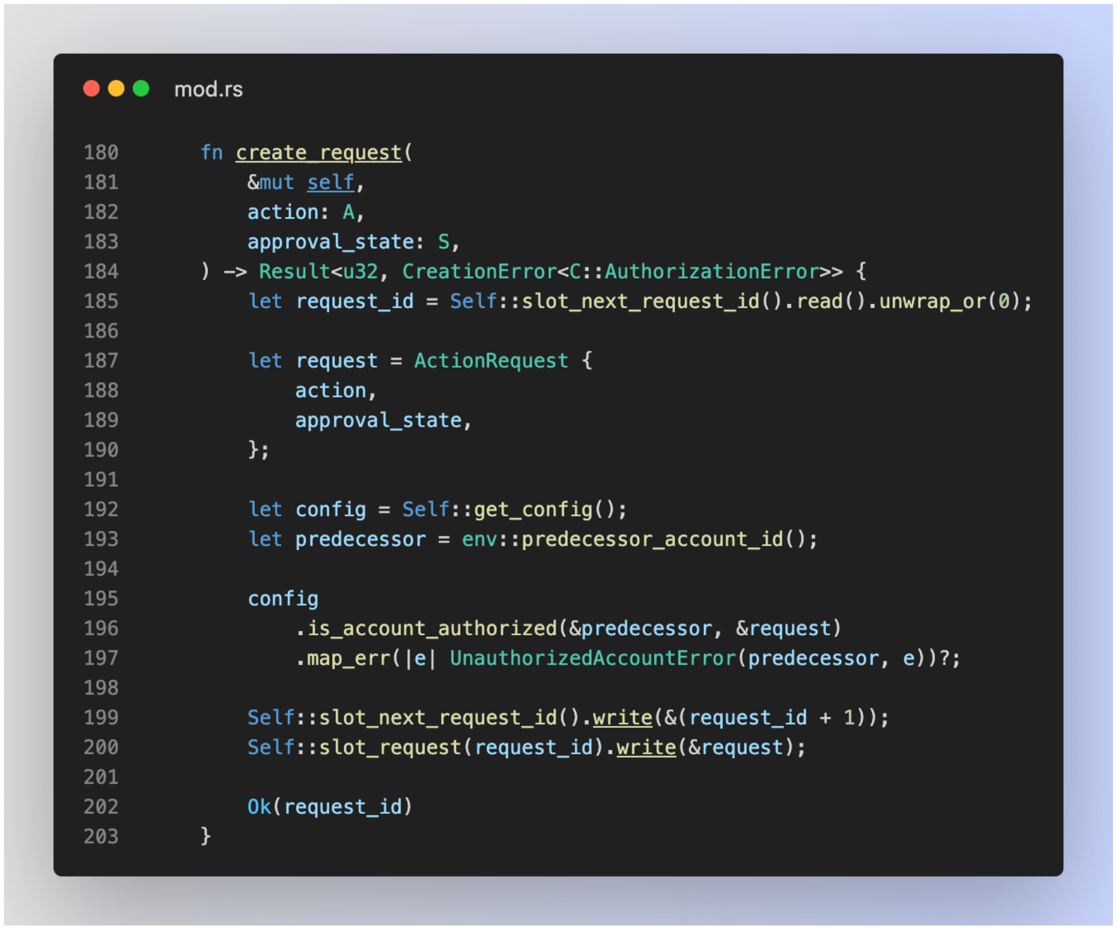
utils.rs

```
39  pub fn apply_storage_fee_and_refund(
40      initial_storage_usage: u64,
41      additional_fees: u128,
42  ) -> Option<Promise> {
43      // Storage consumption after storage event
44      let storage_usage_end = env::storage_usage();
45
46      // Storage fee incurred by storage event, clamped >= 0
47      let storage_fee = u128::from(storage_usage_end.saturating_sub(initial_storage_usage))
48          * env::storage_byte_cost();
49
50      let total_required_deposit = storage_fee + additional_fees;
51
52      let attached_deposit = env::attached_deposit();
53
54      require!(
55          attached_deposit >= total_required_deposit,
56          format!(
57              "Insufficient deposit: attached {attached_deposit} yoctoNEAR < required {total_requi
58          )
59      );
60
61      let refund = attached_deposit - total_required_deposit;
62
63      // Send refund transfer if required
64      if refund > 0 {
65          Some(Promise::new(env::predecessor_account_id()).transfer(refund))
66      } else {
67          None
68      }
69  }
```

Utils.rs line 47 has unchecked multiplication, hopefully storage_byte_cost and the storage used are sufficiently low that an overflow would not occur but checked_mul with validation would prevent overflows.

Similarly, line 50 has unchecked math, which can lead to an overflow and a higher-than-expected refund. If additional fees were exposed to contract callers in other functions, the vulnerability can be directly exploited.

Additionally, an overflow is possible on line 199 of src/approval/mod.rs.


mod.rs

```
180     fn create_request(
181         &mut self,
182         action: A,
183         approval_state: S,
184     ) -> Result<u32, CreationError<C::AuthorizationError>> {
185         let request_id = Self::slot_next_request_id().read().unwrap_or(0);
186
187         let request = ActionRequest {
188             action,
189             approval_state,
190         };
191
192         let config = Self::get_config();
193         let predecessor = env::predecessor_account_id();
194
195         config
196             .is_account_authorized(&predecessor, &request)
197             .map_err(|e| UnauthorizedAccountError(predecessor, e))?;
198
199         Self::slot_next_request_id().write(&(request_id + 1));
200         Self::slot_request(request_id).write(&request);
201
202         Ok(request_id)
203     }
```

Recommendation

Overflow can be checked in release versions by specifying it in Cargo.toml:

```
[profile.release]
overflow-checks = true
```

Alternatively, checked math functions (`checked_add`, `checked_mul`, etc.) can be used.

Remediation

Line 50 was fixed in this commit. Line 47 was accepted due to validation in the NEAR VM. Line 199 accepted because overflow is prohibitive due to gas costs.

KSI-11 – Renaming may remove extra characters

Severity	**INFORMATIONAL**
Status	**Resolved**

Impact	Likelihood	Difficulty
N/A	N/A	N/A

Description
Near Contract Tools uses the Heck library to convert between naming conventions. The Heck documentation explains how some characters may be removed during conversion.

Impact
If Heck removes characters from a string, the NEAR event would be slightly different from the original name. This could be unexpected behavior for a developer expecting the exact name to appear in the event. If some infrastructure outside the blockchain is expecting the original name, the infrastructure might misinterpret or entirely miss the event.

Evidence
“Characters not within words (such as spaces, punctuations, and underscores) are not included in the output string except as they are a part of the case being converted to. Multiple adjacent word boundaries (such as a series of underscores) are folded into one. (“hello__world” in snake case is therefore “hello_world”, not the exact same string). Leading or trailing word boundary indicators are dropped, except insofar as CamelCase capitalizes the first word.”

Recommendation
Warn developers that characters may change during the renaming process in appropriate documentation.

Reference
https://github.com/withoutboats/heck/blob/master/README.md

Remediation
A code comment was added explaining this issue in the repository.

KSI-12 – Missing test coverage

Severity	**INFORMATIONAL**
Status	**Resolved**

Impact	Likelihood	Difficulty
N/A	N/A	N/A

Description
Test coverage for some functions is missing.

Impact
Functionality is not guaranteed, especially with code updates, without proper test coverage.

Evidence

Specifically, `iter_members_of` in `src/rbac.rs` is missing unit tests.

Native_transaction_actions.rs is missing workspace test coverage for the following NEAR actions:

- `CreateAccount`
- `DeployContract`
- `Stake`
- `AddKey`
- `DeleteKey`
- `DeleteAccount`

Utils.rs is missing workspace tests for:

- `prefix_key`
- `apply_storage_fee_and_refund`

Recommendation
Write unit tests for `iter_members_of` as well as workspace tests for the functions listed above.

Remediation

Tests were added to the codebase in the repository prior to and including commit a759b642e904edcbfd9e0d336c35085a03e2edf5.

METHODOLOGY

During this source code review, the Kudelski Security services team reviewed the source code provided by NEAR Foundation. Each team member independently worked to review the code, then reviewed all individual findings together as a team.

Following additional discussion, findings and opinions were merged to create this report.

During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories (where applicable):

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

Tools

- Visual Studio Code
- Semgrep

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system. If any vulnerabilities are detected, they will be identified and scored within the document and will be presented as such:

Impact

The overall effect of the vulnerability on the system or organization is based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

High:
The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client.

Medium:
It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.

Low:
There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge.

High:
It is extremely likely that this vulnerability will be discovered and abused.

Medium:
It is likely that this vulnerability will be discovered and abused by a skilled attacker.

Low:
It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty

Difficulty is measured according to the ease of exploitation by an attacker based on readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

Low:
The vulnerability is easy to exploit or has readily available techniques for exploit.

Medium:
The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.

High:
The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit.

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty.

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Ken Toler	Director – Application Security and Blockchain	ken.toler@kudelskisecurity.com
Rez Khan	Blockchain Security Expert	rez.khan@kudelskisecurity.com
Emily Howson	Blockchain Security Expert	emily.howson@kudelskisecurity.com
Porter Adams	Blockchain Security Expert	porter.adams@kudelskisecurity.com