NEAR

NEAR Protocol

Security Assessment Report

Version: 2.1

June, 2022

Contents

Introduction

NEAR is a blockchain leveraging a proof-of-stake consensus algorithm and a sharding design to address some of the scalability and performance challenges faced by existing blockchain networks.

Sigma Prime performed several security reviews (in Q4 2019, Q1 2020 and Q2 2020) targeting NEARCore (the official reference implementation) and initial smart contracts, powered by the NEAR platform and developed in Rust.

Sigma Prime was commercially engaged to perform a security asssessment of the `w-near` smart contract, `FungibleToken` standard library and the ETH-NEAR Rainbow Bridge.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given, which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*. Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities found within the scope of this review.

Overview

The NEAR contracts can be logically separated into two parts:

- `w-near` contracts representing the implementation of a wrapped version of the `NEAR` token. Users are able to wrap (mint) and unwrap (burn) `NEAR` and `wNEAR` tokens respectively. `near_deposit` and `near_withdraw` methods are used enter and exit the system.
- The implementation of the `FungibleToken` (NEP-141) library which aims to act as a standard interface for fungible tokens. `NEP-141` is similar to `NEP-21` (comparable to Ethereum's `ERC20` standard) but with the addition of the `ft_transfer_call` method. This new method allows the receiver contract to work in concert with the token contracts, enabling a single transaction to both transfer and call a method.

The ETH-NEAR Rainbow Bridge can also be logically separated into several distinct parts:

- The `EthOnNearClient` implementation accepts Ethereum block headers and maintains the canonical chain within a NEAR contract. A finalized threshold is used to correctly track the canonical chain, preventing users from building invalid blocks upon the canonical chain due to chain re-orgs. The client tracks the last 7 days of finalized blocks, ensuring the contract state does not grow endlessly. As a result, users are required to finish their bridge transfers within 7 days, or risk losing their funds.
- The `NearOnEthClient` implementation accepts NEAR block headers and does not need to verify every single NEAR header as long as it verifies at least one header per NEAR epoch. As a result, the `NearOnEthClient` is able to store hashes of all NEAR headers in history. NEAR light clients only accept finalized blocks, so therefore `NearOnEthClient` does not need to deal with chain re-orgs. Signature verification is a costly action on Ethereum, hence NEAR has taken an optimistic approach by allowing anyone to challenge block headers.
- Both client implementations also contain a suite of prover contracts which either verify Ethereum events or NEAR contract execution results.

The ETH-NEAR Rainbow Bridge enables the transfer of tokens from Ethereum to NEAR and vice-versa. The fungible token standard plays an essential role in maintaining compatibility between varying token implementations.

Security Review Summary

This review was initially conducted on the following commits:

- `near-protocol/w-near` : Smart contracts representing the implementation of a wrapped version of the `NEAR` token.
  - Commit 54f8267
- `near-protocol/fungible_token` : Smart contracts representing the implementation for a standard interface for fungible tokens. Aims to be comparable to the `ERC20` standard.
  - Commit bac0ec7
- `aurora-is-near/rainbow-bridge` : The ETH-NEAR Rainbow Bridge, implementations of light clients and prover smart contracts on each protocol.
  - Commit fd7019f
- `aurora-is-near/eth-connector` : Smart contracts representing a two way bridge for sending ETH to the NEAR chain as `NEP-141` represented by `nETH`.
  - Commit 872e643
- `aurora-is-near/near-erc20-connector` : Enables bridging of `NEAR` tokens to Ethereum, represented as `eNEAR` tokens and adhering to the ERC-20 standard.
  - Commit 49cf263
- `aurora-is-near/rainbow-token-connector` : A reference to a generic `ERC-20/NEP-141` connector which uses NEAR's rainbow bridge implementation.
  - Commit 134b3b6

The manual code-review section of the reports, focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the NEAR Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focuses on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- cargo audit: `https://github.com/RustSec/rustsec/tree/main/cargo-audit`
- cargo outdated: `https://github.com/kbknapp/cargo-outdated`
- cargo fuzz: `https://github.com/rust-fuzz/cargo-fuzz`
- cargo hfuzz: `https://github.com/rust-fuzz/honggfuzz-rs`

Output for these automated tools is available upon request.

Fuzzing activities leveraging libfuzzer and honggfuzz have been performed by the testing team in order to identify bugs within the code in scope. `libfuzzer` and `honggfuzz` are coverage-guided tools which explore different code

paths by mutating input to reach as many code paths possible. The aim is to find memory leaks, overflows, index out of bounds or any other unexpected panics. Both tools were used in conjunction as there are certain functions which suited one tool over the other.

Specifically, the testing team produced the following fuzzing targets shared with the development team:

- `fuzz_rlp_decode_blockheader`
- `fuzz_rlp_encode_decode_blockheader`
- `fuzz_rlp_decode_receipt`
- `fuzz_rlp_decode_log_entry`
- `fuzz_rlp_encode_decode_log_entry`
- `fuzz_rlp_decode_blockheader_post_london`
- `fuzz_borsh_blockheader`
- `fuzz_decode_deserialize_blockheader_pre_london`
- `fuzz_borsh_blockheader_post_london`
- `eth_locked_event`
- `eth_unlocked_event`
- `token_metadata_event`
- `eth_transfer_event`
- `eth_deposit_event`

These fuzzing targets have all been shared with the development team as a by-product of this security review. Execution and instrumentation can be done by following the relevant `README.md` file.

Additionally, the team produced a suite of python/rust-based tests. These tests verify some of the core business logic.

The output of these tests are provided in the Appendix (Test Suite), and the implementations have been provided to the development team alongside this report.

Findings Summary

The testing team identified a total of 13 issues during this assessment. Categorized by their severity:

- Low: 5 issues.
- Informational: 8 issues.

All vulnerabilities have been acknowledged and/or resolved by the development team.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the scope of this review. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including comments not directly related to the security posture of the relevant smart contracts, are also described in this section and are labelled as *"informational"*.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team;
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk;
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
NSS-01	Vulnerable Dependencies	Informational	Closed
NSS-02	Incorrect Units in Logs	Informational	Resolved
NSS-03	Unchecked Addition and Subtraction Operations	Informational	Resolved
NSS-04	Lack of Test Coverage	Informational	Resolved
NSS-05	Free Memory Pointer Overflow	Low	Resolved
NSS-06	Borsh Decoding Does Not Call `done()`	Low	Resolved
NSS-07	Hardhat/Truffle Forcefully Compiles Smart Contracts Using An Unsafe Solidity Version	Low	Resolved
NSS-08	`AdminControlled.sol` Lacking Admin Transfer Pattern	Low	Resolved
NSS-09	Gas Supplied To `transfer()` Call Might Change In The Future	Low	Resolved
NSS-10	Users Can Frontrun `challenge()` Function	Informational	Closed
NSS-11	Incorrect Log Output	Informational	Resolved
NSS-12	Requirement for NEP-145 Implementation	Informational	Closed
NSS-13	Miscellaneous General Statements	Informational	Resolved

NSS-01	Vulnerable Dependencies
Asset	`w-near/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

A range of dependencies have been flagged as containing security vulnerabilities on `crates.io`. These are not direct dependencies as they do not appear in `Cargo.toml` but exist in the dependency tree.

The following is a list of crates which are vulnerable:

- `cranelift-codegen` : v0.67.0
- `cranelift-codegen` : v0.68.0
- `generic-array` : v0.12.3
- `raw-cpuid` : v7.0.4

Recommendations

We recommend updating these dependencies to the following versions:

- `cranelift-codegen` : >= v0.73.1
- `generic-array` : >= v0.12.4 OR >= v0.13.3
- `raw-cpuid` : >= v9.0.0

Consider also integrating the tool cargo audit, which identifies any vulnerable dependencies in the dependency tree, into the CI process.

Resolution

Two issues have been created to resolve this issue, the first is to update the dependencies in issue #195. The second, adds `cargo-audit` to the CI process in issue #196. However, wNEAR has been deployed and the bytecode locked on-chain hence, these changes will not be reflected on the mainnet deployment.

NSS-02	Incorrect Units in Logs
Asset	`w_near.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `near_withdraw()` allows a user to withdraw an amount of `NEAR` specified in units of `yoctoNEAR`.

The log emitted states the units as `NEAR` rather than `yoctoNEAR`.

Recommendations

We recommend modifying the logs such that either the units are in `yoctoNEAR` or the amount is in `NEAR`.

Resolution

The issue is resolved in PR #198 to modify the units to say `yoctoNEAR`. However, wNEAR has been deployed and the bytecode locked on-chain hence, these changes will not be reflected on the mainnet deployment.

NSS-03	Unchecked Addition and Subtraction Operations
Asset	`fungible_token/core_impl.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The fungible token standard implementation provides a set of Rust macros. The macros perform unchecked addition, subtraction and multiplication operations. Unchecked mathematical operations will wrap in rust if `overflow_checks` are not turned on in the cargo profile.

This issue is raised as informational as by default, NEAR smart contracts release profile will have `overflow_checks` set to `true`. When set to `true`, any overflows will cause a panic, preventing malicious attacks on the contract.

Recommendations

Consider adding overflow checks to math operations such as addition, multiplication and subtraction within the macro definitions to prevent accidental misuse by developers who wish to have overflow checks turned off for their specific contracts.

Resolution

Pull request #830 has been created to address these issues. It converts all unchecked math operations to checked operations.

NSS-04	Lack of Test Coverage
Asset	`w-near/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Full test coverage is an essential process in ensuring the codebase works as intended (outlined in NEAR's design documentation). Insufficient code coverage may lead to the arisal of unexpected issues due to changes in the underlying smart contract implementation, leading to undetected changes in functionality.

Recommendations

Consider updating the tests for `wNEAR` to include interactions with storage, `ft_transfer_call`, `near_withdraw` and other edge cases specified in the `FungibleToken` standard's documentation.

Resolution

Additional testing has been added in PR #202.

NSS-05	Free Memory Pointer Overflow		
Asset	`Borsh.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Solidity versions prior to `0.6.5` contain a bug that may be exploited in `Borsh.sol` decoding.

The bug exists as an overflow of the memory pointer when allowing arbitrarily length arrays to be passed as input to `public` or `external` functions. The bug stems from Solidity using a linear arena memory allocator. This allocator stores current memory pointers. When memory is to be allocated, the current memory pointer returns the start address of allocated memory. Then the required number of bytes is added to the current memory pointer, wrapping if there is an overflow.

The functions which take `bytes memory data` as input to the function allocate this data in memory through the following code.

```
function decodeArrayAt(uint arg) private returns (bytes memory) {
    uint offset = read(0x04+0x20*arg);
    uint length = read(0x04+offset);
    bytes memory data = malloc(length);
    memcpy(data, 0x04 + offset + 0x20, length);
    return data;
}

function malloc(uint size) private returns (bytes memory) {
    bytes memory buf;
    assembly {
        buf := mload(0x40)
        mstore(0x40, add(add(buf, size), 0x20))
        mstore(buf, size)
    }
    return buf;
}
```

As a result it is possible to overflow the free memory pointer stored at address `0x40` by setting the `length` of an array to a value slightly below 2^{256}. This bug is fixed in future solidity versions by reverting if the memory pointer overflows.

Recommendations

This issue may be mitigated by updating all files to use solidity verion greater than `0.6.5`, however we recommend updating to the most recent solidity version.

Resolution

Each of the libraries using solidity versions prior to `0.8.0` have now been updated as seen in the following pull requests.

- `erc20-connector` has been updated in PR #91
- `eNEAR` has been updated in PR #14
- `eth-custodian` has been updated in PR #37

NSS-06	Borsh Decoding Does Not Call `done()`		
Asset	`eNear.sol`, `ERC20Locker.sol` & `EthCustodian.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Borsh is a serialization protocol that allows encoding data to bytes and decoding data to object orientated structs.

The Borsh protocol expects the exact number of bytes to be parsed when decoding an object to prevent length extension attacks.

In the functions `EthCustodian._decodeBurnResult()`, `ERC20Locker._decodeBurnResult()` and `eNear._decodeBridgeResult()`, borsh data is decoded without performing the final length check, which ensures we have decoded exactly the number of bytes parsed to the decoder.

The implications are that it is possible to append bytes to the data and still have a successful decoding.

Recommendations

Consider executing the function `borshData.done()` after decoding the objects to ensure the correct number of bytes have been parsed.

Resolution

Additional checks have been added to ensure the Borsh decoding uses all of the bytes in the following pull requests.

- `EthCustodian` in PR #30
- `ERC20Locker` in PR #87
- `eNear` in PR #11

NSS-07	Hardhat/Truffle Forcefully Compiles Smart Contracts Using An Unsafe Solidity Version		
Asset	`hardhat-config.js` and `truffle-config.js`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The Hardhat config specifies a solidity version which is incompatible with certain imported and inherited contracts. As a result, running the command, `npx hardhat compile` or `truffle compile`, will result in a number of contracts with Solidity version `^0.8` being forcefully compiled at Solidity version `0.6.12`. Consequently, any overflow/underflow protections provided by `^0.8` will no longer apply to the underlying smart contracts.

Currently, the testing team has not identified affected areas which are exploitable, however, future upgrades to the impacted contracts may introduce unintended exploits.

Recommendations

Consider bumping the solidity version of the following contracts from `^0.6.12` to `^0.8`:

- `EthCustodian.sol`
- `ProofKeeper.sol`
- `eNear.sol`
- `ERC20Locker.sol`

Additionally, it is important to ensure `hardhat.config.js` and `truffle-config.js` are also updated to use the most up-to-date Solidity version providing the latest security protections.

Resolution

Each of the libraries using solidity versions prior to `0.8.0` have now been updated as seen in the following pull requests.

- `EthCustodian.sol` & `ProofKeeper.sol` have been updated in PR #37
- `ERC20Locker.sol` has been updated in PR #91
- `eNear.sol` has been updated in PR #14

NSS-08	`AdminControlled.sol` Lacking Admin Transfer Pattern		
Asset	`rainbow-bridge/contracts/eth/nearbridge/contracts/AdminControlled.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `AdminControlled.sol` contract utilises the `adminSstore()` function to update certain state variables within contracts inheriting `AdminControlled.sol`. However, it is possible that the `onlyAdmin` role may be unintentionally set to the wrong account. As a result, any admin related functions will not be callable.

Recommendations

Consider implementing a two-step process when transferring the `onlyAdmin` role, denoted as the `nominateAdmin()` and `acceptAdmin()` functions. It might be useful to add an explicit `renounceAdmin()` function if the NEAR team decides to remove the ownership of contracts inheriting the `AdminControlled.sol` contract.

Resolution

A safe admin transfer pattern has been implemented in PR #705 and updated in #742.

NSS-09	Gas Supplied To `transfer()` Call Might Change In The Future		
Asset	`rainbow-bridge/contracts/eth/nearbridge/contracts/NearBridge.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `transfer()` call is used to send `ETH` to a target user. The call attaches only `2300` gas and will throw an error if the recipient is unable to receive `ETH`. However, the gas costs are liable to change in the future. Therefore, functions utilising this opcode should assume an attacker has sufficient gas to reenter the contract and implement proper check-effects patterns.

Additionally, the `challenge()` function will be prone to reentrancy attacks if such a change is introduced. This would allow a malicious user to add a block and frontrun themselves if anyone attempts to challenge them, with no loss of funds.

Recommendations

Consider utilising the `call()` opcode over `transfer()` and implement the necessary *checks-effects-interactions* pattern to protect these functions from reentrancy attacks. For example, the `challenge()` function should ensure that `lastValidAt` is set to `zero` before any external calls are made.

Resolution

The recommendation has been implemented in three pull requests, PR #729, PR #41 and PR #747.

NSS-10	Users Can Frontrun `challenge()` Function
Asset	`rainbow-bridge/contracts/eth/nearbridge/contracts/NearBridge.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

NEAR has taken an optimistic approach in maintaining the canonical chain on Ethereum. Watchdog services are tasked with monitoring the addition of NEAR light client blocks, performing the necessary verification off–chain. In the event a watchdog service has found a malformed block, the user will attempt to call `challenge()` on the relevant block, verifying the block on–chain and rewarding the challenger half of the staked ETH while burning the rest.

However, there is no incentive for users to run a watchdog service and therefore contribute to the availability and liveness of the ETH-NEAR rainbow bridge. Any user could avoid running a watchdog service and instead monitor the blockchain for MEV opportunities. Hence, an MEV searcher could frontrun all transactions to `challenge()` a given block.

Recommendations

Consider implementing a commit-reveal scheme when users `challenge()` light client blocks. A user could first submit a hash of the proof in the first action before finally revealing the underlying proof in the second action. The first and second actions should be separated by a finite number of blocks. Between these two actions, the user should have claim over the underlying hash, preventing other users from repeating the challenger's actions.

Resolution

The issue has been closed with the following comment from the development team.

We are aware of this thing. It's actually not that easy to solve like that as just providing a hash is not enough: it's an index so it's easy to predict. Creating complex schemes might be an error-prone approach. This will be improved when we get rid of the optimistic approach on Ethereum side for NEAR client. For now, we know that this deincentivizes running of such relayers, though we consider this somehow as a feature too that could be beneficial for us: we can challenge the transaction and the network is quite congested, we can ensure that our transaction will be immediately mined exactly because of MEV.

NSS-11	Incorrect Log Output
Asset	`rainbow-token-connector/metadata-connector/contracts/ERC20MetadataLogger.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The event `Log` has the field `uint256 timestamp`.

When the log is emitted on line [27–33], the field `timestamp` is set to `block.number`.

Recommendations

Ensure that the `Log` is intended to use the block height (`block.number`) rather than the timestamp (`block.timestamp`).

If block height is desired, consider altering the name of the field `Log.timestamp` to `Log.height`.

Otherwise, if the timestamp is desired use `block.timestamp` instead of `block.number`.

Resolution

The event field has been modified to `block_height` to represent the block height rather than timestamp. The update may be seen in PR #88.

NSS-12	Requirement for NEP-145 Implementation
Asset	`eth-connector/evm-fungible-token/`
Status	**Closed:** See Resolution
Rating	Informational

Description

NEP-145 is an enhancement proposal which allows for the charging of users for the storage they require. Without NEP-145, the contract owner may be paying all of the storage fees for storing the balances of accounts.

As a result, there is a possible DoS attack where a contract may run out of funds. If users create transactions which require a large quantity of storage, the storage cost increases. The storage costs are taken from the contract, which if it is sufficiently high may drain all available funds.

As NEP-145 is still under active development it has not yet been implemented in the contract. Thus, the contract is vulnerable to this issue.

Recommendations

We recommend implementing NEP-145 at the earliest time when it is marked as final.

Alternatively, charge users for storage fees of creating new accounts through other means.

Resolution

The files have been removed in PR #34 and thus the issue is no longer relevant.

NSS-13	Miscellaneous General Statements
Asset	`Smart Contract Suite`
Status	**Resolved:** See Resolution
Rating	Informational

Description

1. **`UNPAUSE_ALL` and `UNPAUSED_ALL` are unused:**

   The files `NearBridge.sol` and `NearProver.sol` both contain the private constant variable `UNPAUSE_ALL = 0`, the files `EthCustodian.sol` and `ECR20Locker.sol` contain the variable `UNPAUSED_ALL`. A private constant is not easily accessible off-chain and since it is not used on-chain it may be safely removed.

2. **Unnecessarily permissive check:**

   On line [206] in `NearBridge.sol` the number of approvals is required to be greater than or equal to the number of producers `nearBlock.approvals_after_next.length >= thisEpoch.numBPs`. However, it iterates through exactly `thisEpoch.numBPs` approvals when calculating the stake and setting the signature set. Consider restricting the check to a strict equality check as excess approvals are not used.

3. **Commented out code:**

   There is commented out code on line [63–34] of `ProofDecoder.sol`. Consider deleting these lines.

4. **Missing SPDX licence identifiers:**

   There are a number of contracts lacking an SPDX licence identifier, generating a warning during compilation. Consider adding an appropriate licence to the affected files.

5. **Mismatch with `amount` input used in `Deposited` and `Withdrawn` events and `BurnResult` struct:**

   The `EthCustodian.sol` contract uses `uint128` for some `amount` inputs and `uint256` in others. Consider updating `uint128` instances of `amount` to `uint256` to protect against overflow. Currently, the `uint128` types provide no additional gas savings and only contribute to potential security issues.

6. **`separator` variable can be made `constant`:**

   The `separator` variable is used within `EthCustodian.sol` when generating the protocol message upon EVM deposits. This can be made constant for some small gas savings.

7. **Slot packing is not optimized:**

   The order of state variables in `NearBridge.sol` isn't optimised for efficient slot packing. Consider ordering the variables such that the number of slots in `NearBridge.sol` is reduced.

8. **Unspecified `uint` types:**

   Consider adhering to Solidity's best code practices by updating instances of `uint` to a specific variable size.

9. **Constructor lacking input validation:**

   `AdminControlled.sol` does not ensure that its constructor arguments are properly validated and hence contracts inheriting `AdminControlled.sol` may be put into an unexpected state. Consider avoiding this by validating the inputs of the contract's constructor.

10. **Gas savings by caching storage accesses:**
    `NearBridge.addLightClientBlock()` performs multiple storage accesses to the `epochs` array. There are potential gas savings if `epochs` is cached once in memory and written into storage only where necessary.
11. **Revert messages are lacking clarity:**
    There are several cases where contracts have a `require()` statement with no error message attached. Users interacting with these functions will be unable to determine what went wrong in their transaction. Consider adding relevant error messages to affected `require()` statements.
12. **Unimplemented `adminReceiveEth()` function:**
    The `adminReceiveEth()` function has the same logic as the default `receive()` function except users have to attach a function signature to their call instead of simply sending `msg.value` to the contract. Consider changing `adminReceiveEth()` to `receive()`.
13. **Inconsistent naming of Light vs Lite:**
    There are inconsistencies between the use of light and lite when referring to a light client. For example, in `ProofDecoder.sol` there is `BlockHeaderLight block_header_lite` on line [15]. We recommend using consistent naming to avoid confusion and accidental misuse.
14. **Inconsistent formatting in Solidity variable naming:**
    It is recommended to use the same formatting guide when naming variables to differentiate between storage variables, local variables and function parameters. For example, the `eth-connector` repository uses a trailing underscore to represent state variables, whereas the `rainbow-bridge` repository uses no underscores. We recommend using one style guide across all repositories for consistency.
15. **Spelling and typos:**
    - `eth-connector/evm-fungible-token/src/lib.rs`
      - line [171] "// TODO: NEP-145 Account Storage impelemtation nee" -> "fee"
      - line [172] "// It spent additonal account amount fot storage" -> "for"
    - `Bridge.sol`, `ProofKeeper.sol` and `Locker.sol`
      - "Proof is from the ancient block" -> "from an ancient block"

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

1. This issue is closed with the following comment from the testing team. *This is needed for visibility on usage.*
2. The development team have decided not to patch this issue.
3. The commented out code is deleted in PR #706.
4. Licenses have been included in PR #92.
5. Since the `uint128` values come from NEAR VM which has a max size of 128 bits and `msg.value` will not overflow when cast down to `uint128` since that is sufficient to hold the entire Ethereum total supply.

6. The recommendations was added in PR #32.
7. Slot packing has been implemented in #727.
8. Closed as `uint` is not expected to be changed from an alias of `uint256` in solidity.
9. Closed as these are superfluous checks that are not necessary at deployment time.
10. Closed as the implementation of storaging cache requires too much complexity.
11. Closed since the revert messages have already been updated except those deliberately left empty.
12. Closed as this is deliberate admin functionality.
13. Closed with the following comment from the development team.

    I agree with that inconsistency. The reason why we do have as it's like that in nearcore 's API. I don't think that fixing it only on the smart-contract level will help. When this is fixed on NEAR side, this will be changed on the smart-contract level too. However, I expect that this will be quite breaking for the NEAR RPC. Created an issue for that anyway, so it's trail is there.
14. The variable naming has been updated in the following PRs #93 and #39.
15. Spelling and typos have been fixed.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `pytest` framework was used to perform these tests and the output is given below.

```
test_intialisation                                    PASSED  [3%]
test_near_deposit                                     PASSED  [6%]
test_near_deposit_zero_amount                         PASSED  [9%]
test_near_deposit_insufficient_storage_balance        PASSED  [12%]
test_near_withdraw                                    PASSED  [15%]
test_near_withdraw_incorrect_attachment               PASSED  [18%]
test_near_withdraw_not_registered                     PASSED  [21%]
test_near_withdraw_insufficient_balance               PASSED  [25%]
test_ft_transfer                                      PASSED  [28%]
test_ft_transfer_without_yocto_attachment             PASSED  [31%]
test_ft_transfer_not_registered                       PASSED  [34%]
test_ft_transfer_zero_amount                          PASSED  [37%]
test_ft_transfer_same_receiver                        PASSED  [40%]
test_ft_transfer_log_memo                             PASSED  [43%]
test_ft_transfer_insufficient_balance                 PASSED  [46%]
test_ft_transfer_call_refund                          PASSED  [50%]
test_ft_transfer_call_promise_fails                   PASSED  [53%]
test_ft_transfer_call_returns_invalid_amount          PASSED  [56%]
test_ft_transfer_call_send_tokens_back_to_sender      PASSED  [59%]
test_ft_transfer_call_unregister_sender               PASSED  [62%]
test_ft_transfer_call_unregister_receiver             PASSED  [65%]
test_ft_transfer_call_without_yocto_attachment        PASSED  [68%]
test_storage_deposit                                  PASSED  [71%]
test_storage_deposit_insufficient_balance             PASSED  [75%]
test_storage_withdraw                                 PASSED  [78%]
test_storage_withdraw_more_than_available_balance     PASSED  [81%]
test_storage_withdraw_not_registered                  PASSED  [84%]
test_storage_withdraw_incorrect_attachment            PASSED  [87%]
test_storage_unregister                               PASSED  [90%]
test_storage_unregister_force                         PASSED  [93%]
test_storage_unregister_not_registered                PASSED  [96%]
test_storage_unregister_with_balance                  PASSED  [100%]
```

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_only_admin                                                PASSED  [1%]
test_admin_send_eth                                            PASSED  [2%]
test_paused                                                    PASSED  [4%]
test_delegatecall                                              PASSED  [5%]
test_deploy                                                    PASSED  [6%]
test_finalize_near_to_eth_transfer                             PASSED  [8%]
test_finalize_near_to_eth_transfer_withdraw_flag               PASSED  [9%]
test_finalize_near_to_eth_transfer_bad_proof                   PASSED  [10%]
test_finalize_near_to_eth_transfer_height                      PASSED  [12%]
test_finalize_near_to_eth_transfer_receipt_replay              PASSED  [13%]
test_finalize_near_to_eth_transfer_executor_id                 PASSED  [14%]
test_finalize_near_to_eth_transfer_status_failed               PASSED  [16%]
test_finalize_near_to_eth_transfer_status_unknown              PASSED  [17%]
test_transfer_to_near                                          PASSED  [18%]
test_sha512                                                    PASSED  [20%]
test_check_valid_signatures                                    PASSED  [21%]
test_check_signature_curve_group_order                         PASSED  [22%]
test_check_invalid_signatures                                  PASSED  [24%]
test_deploy                                                    PASSED  [25%]
test_lock_token                                                PASSED  [27%]
test_lock_token_limit                                          PASSED  [28%]
test_lock_token_safe_transfer_from                             PASSED  [29%]
test_unlock_token                                              PASSED  [31%]
test_unlock_token_withdraw_flag                                PASSED  [32%]
test_unlock_token_borsh_burn_result                            PASSED  [33%]
test_unlock_token_min_block_height                             PASSED  [35%]
test_unlock_token_bad_proof                                    PASSED  [36%]
test_unlock_token_status_failed                                PASSED  [37%]
test_unlock_token_status_uknown                                PASSED  [39%]
test_unlock_token_executor_id                                  PASSED  [40%]
test_token_fallback                                            PASSED  [41%]
test_admin_transfer                                            PASSED  [43%]
test_admin_transfer_not_admin                                  PASSED  [44%]
test_deploy                                                    PASSED  [45%]
test_deploy_bad_near_account                                   PASSED  [47%]
test_depoist_to_evm                                            PASSED  [48%]
test_depoist_to_evm_insufficient_value                         PASSED  [50%]
test_depoist_to_near                                           PASSED  [51%]
test_depoist_to_near_insufficient_value                        PASSED  [52%]
test_withdraw                                                  PASSED  [54%]
test_withdraw_invalid_executor                                 PASSED  [55%]
test_withdraw_invalid_eth_custodian                            PASSED  [56%]
test_withdraw_reuse_receipt                                    PASSED  [58%]
test_withdraw_min_block_height                                 PASSED  [59%]
test_withdraw_status_failed                                    PASSED  [60%]
test_withdraw_status_unknown                                   PASSED  [62%]
test_withdraw_invalid_proof                                    PASSED  [63%]
test_deploy                                                    PASSED  [64%]
test_deposit                                                   PASSED  [66%]
test_deposit_invalid_amount                                    PASSED  [67%]
test_deposit_existing_balance                                  PASSED  [68%]
test_withdraw                                                  PASSED  [70%]
test_withdraw_no_balance                                       PASSED  [71%]
```

```
test_withdraw_last_submitter                          PASSED  [72%]
test_init                                             PASSED  [74%]
test_init_wrong_order                                 PASSED  [75%]
test_init_when_initialized                            PASSED  [77%]
test_initWithValidators_wrong_stage                   PASSED  [78%]
test_addLightClientBlock                              PASSED  [79%]
test_addLightClientBlock_invalid_epoch                PASSED  [81%]
test_addLightClientBlock_insufficient_balance         PASSED  [82%]
test_addLightClientBlock_replace_duration             PASSED  [83%]
test_addLightClientBlock_timestamp_overflow           PASSED  [85%]
test_addLightClientBlock_current_height               PASSED  [86%]
test_addLightClientBlock_insufficient_num_approvals   PASSED  [87%]
test_addLightClientBlock_insufficient_stake_approvals PASSED  [89%]
test_addLightClientBlock_new_epoch_without_next_bps   PASSED  [90%]
test_addLightClientBlock_different_next_bps_hash      PASSED  [91%]
test_addLightClientBlock_not_initialized              PASSED  [93%]
test_challenge                                        PASSED  [94%]
test_challenge_valid_signature                        PASSED  [95%]
test_challenge_no_signature                           PASSED  [97%]
test_challenge_last_valid_at                          PASSED  [98%]
test_pausing                                          PASSED  [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

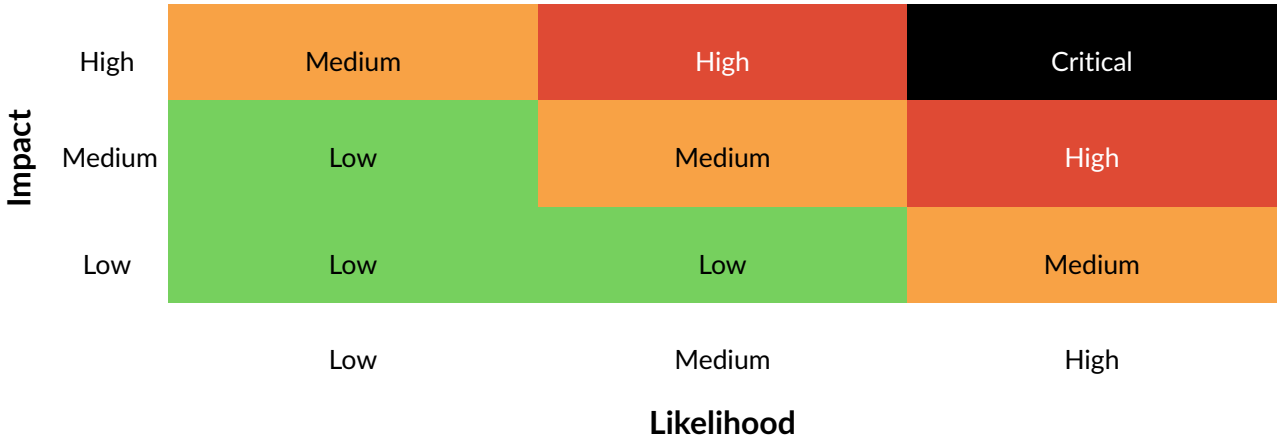

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].