AUDITONE

Audit Report

AURORA

Engine/Bridge

10.05.2024

Table of Contents

CONTENTS

INTRODUCTION

Smart Contract Security Analysis Report

Note: This report may contain sensitive information on potential vulnerabilities and exploitation methods. This must be referred internally and should be only made available to the public after issues are resolved (to be confirmed prior by the client and AuditOne).

INTRODUCTION

Defsec, Csanuragjain, Aktech297 and Ubermensch3dot0, who are auditors at AuditOne, successfully audited the smart contracts (as indicated below) of Aurora Engine. The audit has been performed using manual analysis. This report presents all the findings regarding the audit performed on the customer's smart contracts. The report outlines how potential security risks are evaluated. Recommendations on quality assurance and security standards are provided in the report.

01-PROJECT DESCRIPTION

The Aurora environment consists of the Aurora Engine, a high performance EVM—Ethereum Virtual Machine—and the Rainbow Bridge, facilitating trustless transfer of ETH and ERC-20 tokens between Ethereum and Aurora, within a great user experience.

Aurora exists and is operated as an independent, self-funded initiative, but will continue to leverage the shared team DNA and continually evolving technology of the NEAR Protocol.

The governance of Aurora will take a hybrid form of a Decentralized Autonomous Organization—the AuroraDAO—complemented by a traditional entity which will hold one of several seats in the AuroraDAO.

This audit focused on the Fast Bridge, one-way semi-decentralized bridge created to speed up transfers from Near to Ethereum.

02-Project and Audit Information

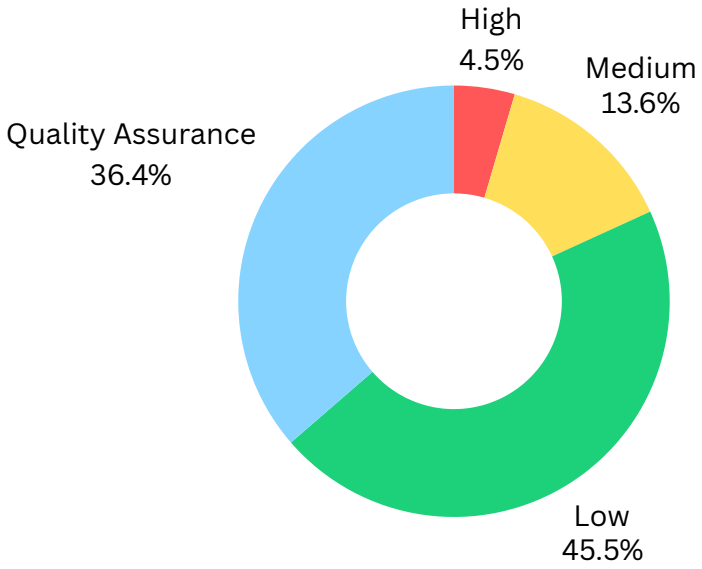
Term	Description
Auditor	Defsec, Csanuragjain, Aktech297 and Ubermensch3dot0
Reviewed by	Luis Buendia and Gracious
Type	Engine/Bridge
Language	Rust
Ecosystem	Near
Methods	Manual Review
Repository	https://github.com/aurora-is-near/aurora-engine
Commit hash (at audit start)	1213f2c7c035aa523601fced8f75bef61b4728ab
Commit hash (after resolution)	987d8381a149154cf6c2ba4977a69af1daf859d5
Documentation	[Added once the whitepaper is published by the project]
Unit Testing	No
Website	https://aurora.dev/
Submission date	14-11-2023
Finishing date	10-05-2024

03-Contracts in Scope

- engine/src/contract_methods/connector/admin_controlled.rs
- engine/src/contract_methods/connector/deposit_event.rs
- engine/src/contract_methods/connector/errors.rs
- engine/src/contract_methods/connector/external.rs
- engine/src/contract_methods/connector/fungible_token.rs
- engine/src/contract_methods/connector/internal.rs
- engine/src/contract_methods/connector/mod.rs
- engine/src/contract_methods/silo/mod.rs
- engine/src/contract_methods/silo/whitelist.rs
- engine/src/contract_methods/admin.rs
- engine/src/contract_methods/evm_transaction.rs
- engine/src/contract_methods/mod.rs
- engine/src/contract_methods/xcc.rs
- engine/src/accounting.rs
- engine/src/engine.rs
- engine/src/errors.rs
- engine/src/hashchain.rs
- engine/src/lib.rs
- engine/src/map.rs
- engine/src/pausable.rs
- engine/src/prelude.rs
- engine/src/state.rs
- engine/src/xcc.rs

04-Executive summary

Aurora Engine smart contracts were audited between 14-11-2023 and 28-04-2024 by defsec, ak1, csanuragjain and ubermensch. Manual analysis was carried out on the code base provided by the client. The following findings were reported to the client. For more details, refer to the findings section of the report.



Issue Category	Issues Found	Resolved	Acknowledged
High	1	1	0
Medium	3	3	0
Low	10	7	3
Quality Assurance	8	3	5

05-Severity Definitions

Risk factor matrix	Low	Medium	High
Occasional	L	M	H
Probable	L	M	H
Frequent	M	H	H

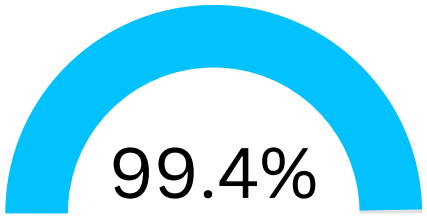
High: Funds or control of the contracts might be compromised directly. Data could be manipulated. We recommend fixing high issues with priority as they can lead to severe losses.

Medium: The impact of medium issues is less critical than high, but still probable with considerable damage. The protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions.

Low: Low issues impose a small risk on the project. Although the impact is not estimated to be significant, we recommend fixing them on a long-term horizon. Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

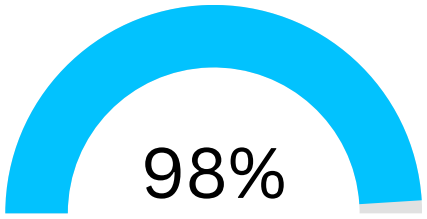
Quality Assurance: Informational and Optimization - Depending on the chain, performance issues can lead to slower execution or higher gas fees. For example, code style, clarity, syntax, versioning, off-chain monitoring (events etc.)

06-Audit Overview



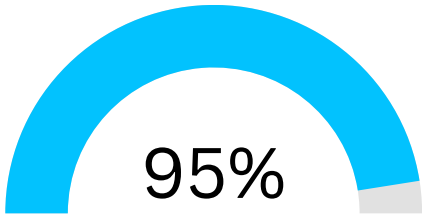
Security score

Security score is a numerical value generated based on the vulnerabilities in smart contracts. The score indicates the contract's security level and a higher score implies a lower risk of vulnerability.



Code quality

Code quality refers to adherence to standard practices, guidelines, and conventions when writing computer code. A high-quality codebase is easy to understand, maintain, and extend, while a low-quality codebase is hard to read and modify.



Documentation quality

Documentation quality refers to the accuracy, completeness, and clarity of the documentation accompanying the code. High-quality documentation helps auditors to understand business logic in code well, while low-quality documentation can lead to confusion and mistakes.

07-Findings

Finding: #1

Issue: Potential Permanent Token Lock Due to **nearProofProducerAccount** Change in Migration Process

Severity: High

Where: eth-connector/src/lib.rs#L325-L349
eth-connector/src/lib.rs#L325-L349

Impact: Users may experience permanent loss of tokens due to the inability to validate pre-migration withdrawal proofs following the migration. This occurs because withdrawal proofs generated by the old connector cannot be validated against the new **nearProofProducerAccount**.

Description: In the Aurora Engine's bridge mechanism, users deposit tokens on Ethereum and claim them on NEAR (and vice versa) by submitting a proof of the deposit, and they withdraw the same way by providing a proof of withdrawal on the other side. During migration, there will be a change in the **nearProofProducerAccount** variable on the Ethereum side to point to the new connector. However, if there are pending withdrawals processed by the engine's connector before the pausing transaction is executed, these transactions will result in tokens being locked permanently. This lock occurs because the Ethereum side will fail to validate proofs against the new **nearProofProducerAccount**, as these proofs were generated by the old connector.

Recommendations: Consider performing the pausing of the engine on multiple steps by pausing the withdrawals first, settling them on the ethereum side, then pausing the engine's other functionalities to proceed with the migration to avoid any race conditions.

Status: Resolved.

Finding: #2

Issue: Potential State Inconsistency Due to **ft_resolve_transfer** Failures During Pausing

Severity: Medium

Where: https://github.com/aurora-is-near/aurora-engine/blob/1213f2c7c035aa523601fced8f75bef61b4728ab/engine/src/contract_methods/connector/internal.rs#L246-L252

Impact: This issue could lead to a state where legitimate transfers are mistakenly reverted when the system is paused, resulting in incorrect or inconsistent states being migrated to the new connector.

Description: Before migration, the Aurora Engine, especially transfers, deposits, and withdrawals, will be paused. However, this pausing mechanism could cause failures in **ft_resolve_transfer** calls. These failures arise because **ft_resolve_transfer** includes an initial check requiring the system to be unpaused. If the system is paused, this function could erroneously revert legitimate transfers. Consequently, the incorrect state may be transferred during migration, leading to potential data integrity issues.

Recommendations: Consider adjusting the **ft_resolve_transfer** function to allow it to execute during the paused state, ensuring that legitimate transfers are not adversely affected during the pause.

Status: Resolved.

Finding: #3

Issue: Engine Methods Accessibility - Restrict to Engine Account Only, Not Owner

Severity: Medium

Where: eth-connector/src/lib.rs#L252C1-L293C1

Impact: Allowing the contract owner to access engine-specific methods can pose a security risk, as it centralizes control and may lead to unintended actions or misuse.

Description: The current implementation of **engine_ft_transfer** and **engine_ft_transfer_call** methods in the **EthConnectorContract** allows both the designated engine account and the contract owner to access these methods. This approach deviates from the standard practice where engine methods should be exclusively accessible by the engine account for operational security and clarity.
Both engine account and contract owner have access rights to the engine methods (**engine_ft_transfer** and **engine_ft_transfer_call**).

```rust
/// Check access right for predecessor account
///
/// # Errors
///
/// Error is returned if the caller hasn't rights.
fn assert_access_right(&self) -> Result<(), error::AdminControlledError> {
    if self.get_account_with_access_right() == near_sdk::env::predecessor_account_id()
        || self.is_owner()
    {
        Ok(())
    } else {
        Err(error::AdminControlledError::AccessRight)
    }
}
```

Recommendations: Modify the **assert_access_right** function to ensure that only the engine account can invoke the engine methods. Remove the condition that allows the contract owner to access these methods.

Status: Resolved.

Finding: #4

Issue: connector.rs : engine compatible method can not be compiled properly due to incorrect typo in the derived attributes.

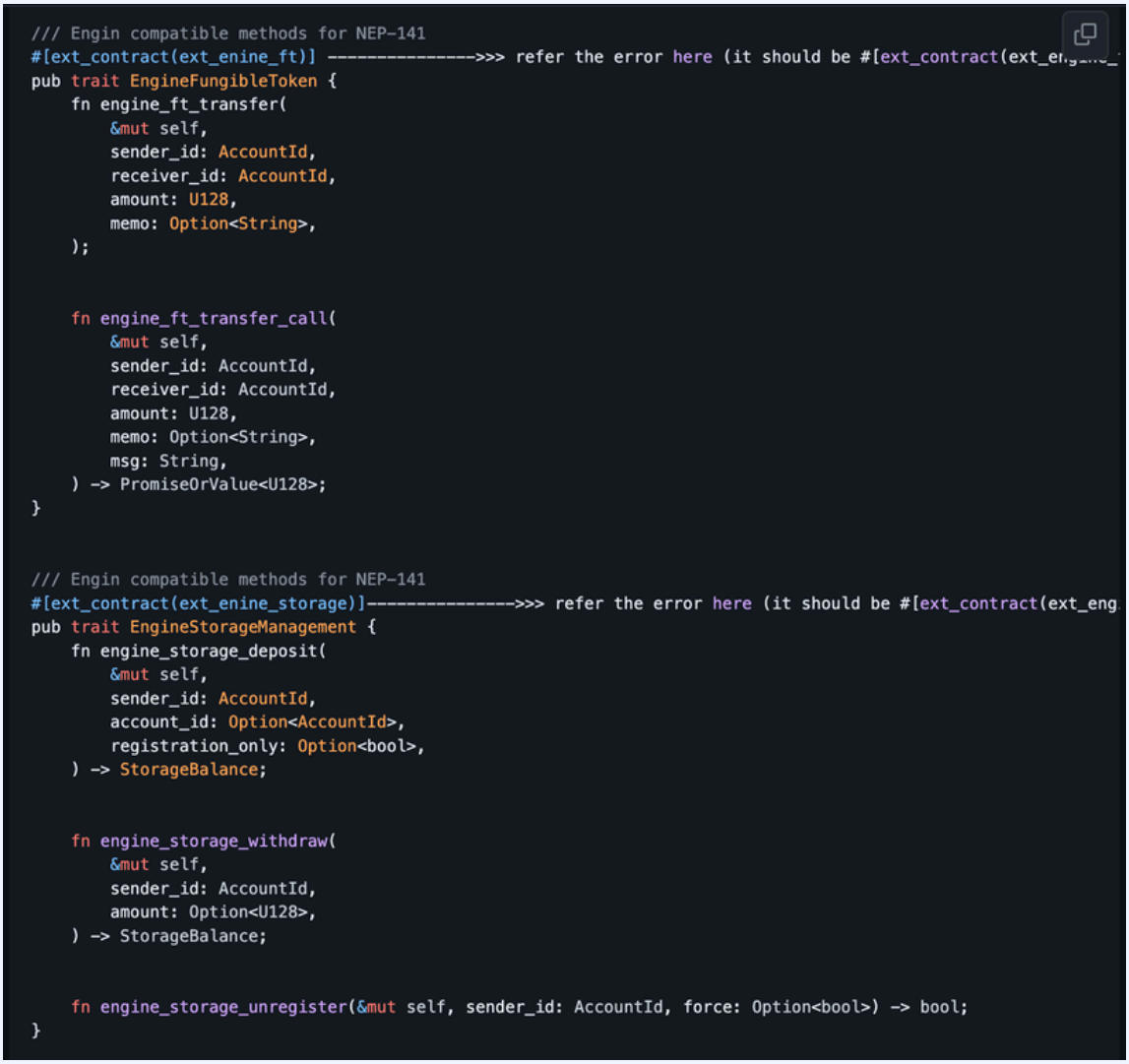
Severity: Medium

Where: aurora-eth-connector/blob/master/eth-connector/src/connector.rs

Impact: The methods or struct which are implementing this **EngineFungibleToke**n and **EngineStorageManagement** can not be properly compiled.

Description: The connector contract has certain traits such as EngineFungibleToken and EngineStorageManagement . These have decorated with attributes such as **#[ext_contract(ext_enine_storage)]**. In rust smart contract the like this are often used to provide additional information or instructions to the underlying blockchain runtime or compiler.

eth-connector/src/connector.rs#L52-L90



```
/// Engin compatible methods for NEP-141
#[ext_contract(ext_enine_ft)] ---------------->>> refer the error here (it should be #[ext_contract(ext_e
pub trait EngineFungibleToken {
    fn engine_ft_transfer(
        &mut self,
        sender_id: AccountId,
        receiver_id: AccountId,
        amount: U128,
        memo: Option<String>,
    );


    fn engine_ft_transfer_call(
        &mut self,
        sender_id: AccountId,
        receiver_id: AccountId,
        amount: U128,
        memo: Option<String>,
        msg: String,
    ) -> PromiseOrValue<U128>;
}


/// Engin compatible methods for NEP-141
#[ext_contract(ext_enine_storage)]---------------->>> refer the error here (it should be #[ext_contract(ext_eng
pub trait EngineStorageManagement {
    fn engine_storage_deposit(
        &mut self,
        sender_id: AccountId,
        account_id: Option<AccountId>,
        registration_only: Option<bool>,
    ) -> StorageBalance;


    fn engine_storage_withdraw(
        &mut self,
        sender_id: AccountId,
        amount: Option<U128>,
    ) -> StorageBalance;


    fn engine_storage_unregister(&mut self, sender_id: AccountId, force: Option<bool>) -> bool;
}
```

Recommendations: Update the code as shown above to fix this issue.

Status: Resolved.

Finding: #5

Issue: Loss of Event Backward Compatibility in Migration Process

Severity: Low

Where:

Impact: Applications relying on events emitted by the original engine's connector will fail to capture these events post-migration, leading to potential loss of functionality and miscommunication between the engine and dependent applications.

Description: In the migration process, while there is an effort to maintain backward compatibility by forwarding calls from the engine's connector to the new connector, this compatibility does not extend to event emissions. Applications that are designed to listen for events emitted by the engine's connector will no longer receive these notifications because, post-migration, relevant events (like transfer events) will be emitted by the new connector. This discrepancy results in a breakdown of the event-notification mechanism, impacting applications that depend on these events for their operations.

Recommendations: Implement callback functions in the engine which will be called back by the new connector after forwarding any action that emits relevant events. This callback should ensure that events are also emitted at the engine's level.

Status: Acknowledged.

Finding: #6

Issue: Storage deposit for new account can be skipped

Severity: Low

Where: https://github.com/aurora-is-near/aurora-eth-connector/blob/master/eth-connector/src/lib.rs#L95

Impact: Account can skip the storage deposit required during account creation.

Description:

- All new accounts are enforced with a small storage deposit which is taken via **engine_storage_deposit** function.

```
else {
            let min_balance = self.storage_balance_bounds().min.0;
            if amount < min_balance {
                env::panic_str("The attached deposit is less than the minimum storage balance");
            }

            self.ft.internal_register_account(&account_id);
```

- Functions like **ft_transfer**, bridge deposit etc call **register_if_not_exists** if account does not exist.
- **register_if_not_exists** function, directly creates account without collecting any storage deposit

```
fn register_if_not_exists(&mut self, account: &AccountId) {
        if !self.ft.accounts.contains_key(account) {
            self.ft.internal_register_account(account);
        }
    }
```

- Deposit Bridge calls

Recommendations: Minimum deposit should be enforced if account is not registered as done in **engine_storage_deposit** function

Status: Resolved.

Finding: #7

Issue: Vulnerability in Out-of-Date Dependencies

Severity: Low

Where: Cargo.toml#L28, Cargo.toml#L24

Impact: Continued use of these outdated and vulnerable dependencies exposes the project to various security risks, which may compromise the integrity and security of the application.

Description: The project relies on several outdated dependencies with known vulnerabilities. Notably, the **ed25519-dalek, wee_alloc, atty, and borsh** crates have been identified as having security issues. The specific vulnerabilities are as follows:

near 4.1

ed25519-dalek (1.0.1):
Vulnerability: Double Public Key Signing Function Oracle Attack.
Advisory: RUSTSEC-2022-0093.
Solution: Upgrade to version >=2.

wee_alloc (0.4.5):
Vulnerability: Unmaintained crate.
Advisory: RUSTSEC-2022-0054.

atty (0.2.14):
Vulnerability: Potential unaligned read.
Advisory: RUSTSEC-2021-0145.

borsh (0.9.3 & 0.10.3):
Vulnerability: Parsing borsh messages with ZST which are not-copy/clone is unsound.
Advisory: RUSTSEC-2023-0033.

Recommendations: Prioritize updating the affected crates to the latest, secure versions as recommended in the advisorie

Status: Resolved

Finding: #8

Issue: Lack of upgrade function

Severity: Low

Where: aurora-eth-connector

Impact: Without an upgrade function, the contract cannot be upgraded and contract states cannot be migrated

Description: Contracts may need the upgrade function. But, the mechanism does not exist on the contract.

Recommendations: Consider implementing upgrade function.

Status: Resolved

Finding: #9

Issue: Inconsistency in internal_storage_unregister Function Compared to Reference Implementation

Severity: Low

Where: eth-connector/src/lib.rs#L337

Impact: The absence of **saturating_add** in our implementation may lead to an overflow risk, especially in cases where the addition exceeds the data type's maximum limit.

Description: The **internal_storage_unregister** function in the smart contract implementation shows a deviation from the reference implementation in terms of handling balance transfers. The implementation does not utilize **saturating_add** for adding a buffer amount to the minimum storage balance. In contrast, the reference implementation employs **saturating_add**, which is a safer method to prevent possible integer overflow. Reference Implementation : https://github.com/near/near-sdk-rs/blob/master/near-contract-standards/src/fungible_token/storage_impl.rs#L8C1-L32C6

Recommendations: To align with the reference implementation and enhance the safety of the contract, it is recommended to implement **saturating_add** in the **internal_storage_unregister** function. This change will protect against overflow and standardize our contract's approach with established best practices.

Status: Resolved

Finding: #10

Issue: Lack of two-factor authentication

Severity: Low

Where: All Privileged functions

Impact: 2FA will provide additional protection for the compromise of the account.

Description: Privileged functions should check whether one yocto NEAR is attached. This will enable the 2FA in the NEAR wallet for security concerns.
This can be implemented in the contract by adding **assert_one_yocto**, which is recommended for all privileged functions.

Recommendations: Consider using **assert_one_yocto** for 2FA authentication on the privileged functions.

Status: Resolved

Finding: #11

Issue: Efficient Handling of Large Data Sets in Accounts and Proofs Processing

Severity: Low

Where: parser.rs#L36

Impact: Processing large data sets without efficient data handling mechanisms can lead to several issues including:

- Increased memory usage, potentially leading to memory exhaustion.
- Slower performance due to inefficient data processing

Description: The current implementation of the parsing function in the codebase processes potentially large data sets, particularly in the handling of accounts and proofs. There can be concern that this approach may not efficiently manage these large data sets, potentially leading to memory or performance issues. The handling of such large volumes of data without proper mechanisms like chunk processing or pagination can cause significant strain on system resources.

Recommendations: To address this issue, it is recommended to implement chunk-based processing or pagination for large data sets.

Status: Resolved

Finding: #12

Issue: remove_relayer_key is not refunding the 100 tokens that has charged when **add_relayer_key.**

Severity: Low

Where: engine/src/contract_methods/admin.rs#L323-L326
engine/src/contract_methods/admin.rs#L351

Impact: Relayer will incur loss of their deposit

Description: The function add_relayer_key allows to add relayer key with 100 tokens.
After some time, due to some reasons, the relayer key would be removed by calling the function remove_relayer_key

```rust
pub fn remove_relayer_key<I: IO + Copy, E: Env, H: PromiseHandler>(
    io: I,
    env: &E,
    handler: &mut H,
) -> Result<(), ContractError> {
    with_hashchain(io, env, function_name!(), |mut io| {
        let state = state::get_state(&io)?;


        require_running(&state)?;
        require_key_manager_only(&state, &env.predecessor_account_id())?;


        let args: RelayerKeyArgs = serde_json::from_slice(&io.read_input().to_vec())
            .map_err(|_| errors::ERR_JSON_DESERIALIZE)?;


        engine::remove_function_call_key(&mut io, &args.public_key)?;


        let action = PromiseAction::DeleteKey {
            public_key: args.public_key,
        };
        let promise = PromiseBatchAction {
            target_account_id: env.current_account_id(),
            actions: vec![action],
        };


        let promise_id = unsafe { handler.promise_create_batch(&promise) };
        handler.promise_return(promise_id);
```

when we look at the above method ,there are not place to refund the 100 tokens that been charged earlier.

Recommendations: Update the function to refund the 100 tokens to the valid relayer.

Status: Acknowledged.

Finding: #13

Issue: Lack of Pause Check

Severity: Low

Where: src/lib.rs#L394

Impact: If there's an emergency pause mechanism in place for the contract (indicated by checks in other functions like withdraw), not having a similar check in **engine_storage_withdraw** means that this function could still be called even when other parts of the contract are paused.

Description: Whether the absence of a pause check in **engine_storage_withdraw** is an issue depends on the overall design and security model of the contract. If a pause mechanism is part of the contract's emergency response strategy, it would be prudent to ensure that this function adheres to the same standards and practices as the rest of the contract.

Recommendations: If the contract is designed with a pause mechanism for emergencies, ensure that all functions which should be paused in such scenarios are consistently checking this condition.

Status: Resolved

Finding: #14

Issue: Migrate can be called multiple times

Severity: Low

Where: https://github.com/aurora-is-near/aurora-eth-connector/blob/master/eth-connector/src/lib.rs#L595

Impact: Migrate function should be callable only once but due to absence of such checks, it could be called in future bringing centralization risk

Description:

- Observe the migrate function at https://github.com/aurora-is-near/aurora-eth-connector/blob/master/eth-connector/src/lib.rs#L595
- Notice that even after migrate is called successfully, it could be recalled.

Recommendations: Add a new variable **migrateComplete** which turns true when migrate function is success. Also if **migrateComplete** is true then error should be thrown

Status: Acknowledged

Finding: #15

Issue: Inclusion of Failed Transactions in Migration Data Leading to Potential Denial of Service.

Severity: Quality Assurance

Where:
blob/1d33c7245a7bc37cd7a4312517dd906570574308/src/rpc.rs#L328-L365

blob/1d33c7245a7bc37cd7a4312517dd906570574308/src/rpc.rs#L196-L289

Impact: Unexpected delays and additional cost and potentially denial-of-service attack during the migration process.

Description: The migration tool parses transactions interacting with the AURORA engine contract to compile a list of accounts with potential positive balances and gather proofs to prevent post-migration replay attacks. However, the tool does not discriminate based on the transaction status, treating failed transactions as valid. This oversight allows an attacker to flood the system with intentionally failed transactions that, despite failing, will be included in the migration dataset. Consequently, this could significantly inflate the size of the account array with zero-balance accounts, potentially overwhelming the migration process, causing delays, incurring unnecessary costs, and potentially denial-of-service.

Recommendations: Adjust the migration tool to include transaction status checks. Only successful transactions should contribute to the list of accounts and proofs for migration.

Status: Acknowledged.

Finding: #16

Issue: Inability to Unregister and Withdraw Old Storage Deposits Post-Migration

Severity: Quality Assurance

Where: engine/src/contract_methods/connector/mod.rs#L258-L269

engine/src/contract_methods/connector/internal.rs#L296-L316

engine/src/contract_methods/connector/external.rs#L493-L501

eth-connector/src/lib.rs#L420-L423

Impact: Users who have made storage deposits pre-migration may lose the ability to unregister and withdraw these deposits after migration, leading to potential loss of funds.

Description: During the migration process, storage deposits intended for user account storage are not migrated to the new connector. As a result, registered users might be unable to unregister and reclaim their storage deposits. The storage_unregister function in the engine calls **engine_storage_unregister** in the connector, but the new connector will lack the necessary funds to cover these costs due to the absence of migrated storage deposits. This oversight could lead to a situation where users' deposits become inaccessible, potentially causing financial losses and eroding user trust.

Recommendations: Consider transferring the storage deposits to the new connector, this can be estimated knowing the number of registered accounts and the cost per account.

Status: Acknowledged.

Finding: #17

Issue: Custom Access Control Implementation in Smart Contract

Severity: Informational

Where: admin_controlled.rs#L1C1-L77C1

Impact: Custom implementations may inadvertently introduce security vulnerabilities that are less likely in standardized solutions.

Description: The smart contract contains a custom implementation for access control, including pause functionality (PAUSE_DEPOSIT, PAUSE_WITHDRAW) and owner access rights. While custom implementations provide flexibility, they can introduce complexity and potential security risks if not carefully designed and tested.
Using established Aurora Near plugins, can offer more security and maintainability

Recommendations: Replace the custom access control logic with Aurora near plugins.

Status: Resolved

Finding: #18

Issue: Robust Handling of Invalid Account Data in **KeyType::Accounts** Parsing

Severity: Quality Assurance

Where: parser.rs#L72C1-L72C1

Impact: Failing to properly handle invalid account data can result in inconsistencies in the parsed data set, leading to incorrect or incomplete information being processed or stored.

Description: In the current implementation of the **KeyType::Accounts(value)** match arm within the parse function, the code attempts to parse a string into an **AccountId** using **AccountId::from_str(account_str)**. If this parsing fails, the function simply prints a message indicating the failure and continues execution without further action. This approach can potentially lead to the silent ignoring of invalid account data, which might be critical for the migrator's integrity.

Recommendations: It is recommended to implement a more robust error handling mechanism for this scenario. Rather than just logging the error and continuing, the application should either halt execution with an appropriate error message or implement a strategy to manage these errors effectively.

Status: Acknowledged.

Finding: #19

Issue: aurora-eth-connector : function **internal_ft_transfer_call** is not validating the sender balance when **sender != receiver.**

Severity: Quality Assurance

Where: [eth-connector/src/lib.rs#L127-L140]

Impact: The function **internal_ft_transfer_call** calls the balance transfer without checking whether the sender has enough balance or not .

Description: The function internal_ft_transfer_call has the following logic to transfer the funds from sender to receiver.
[eth-connector/src/lib.rs#L127-L140]

```
if sender_id == receiver_id {
    // If `sender_id == receiver_id` we should verify
    // that sender account has sufficient account balance.
    // NOTE: Related to Audit AUR-11 report issue
    require!(
        amount > 0,
        "The amount should be a positive non zero number"
    );
    let balance = self.ft.ft_balance_of(sender_id.clone());
    require!(balance.0 >= amount, "Insufficient sender balance");
} else {
    self.ft
        .internal_transfer(&sender_id, &receiver_id, amount, memo);
}
```

The check **require!(balance.0 >= amount, "Insufficient sender balance");** is done only for **sender == receiver.**
But the check need to be done for both cases.

Recommendations: Pull the following line of codes outside the if block. (top)

```
let balance = self.ft.ft_balance_of(sender_id.clone());
require!(balance.0 >= amount, "Insufficient sender balance");
```

Status: Resolved

Finding: #20

Issue: fungible_token.rs : the function **ft_transfer_call** construct more gas for **ft_on_transfer_call**

Severity: Quality Assurance

Where: https://github.com/aurora-is-near/aurora-engine/blob/1213f2c7c035aa523601fced8f75bef61b4728ab/engine/src/contract_methods/connector/fungible_token.rs#L244C13-L250

Impact: more gas would be forwarded for **ft_on_transfer_call**. This would be used to craft the extra call which could be malicious to perform malicious operation.

Description: Following gas values would be used for **resolve_transfer** and **ft_on_transfer**

```
/// Gas for resolve_transfer: 5 TGas
const GAS_FOR_RESOLVE_TRANSFER: NearGas =
NearGas::new(5_000_000_000_000);

/// Gas for ft_on_transfer
const GAS_FOR_FT_TRANSFER_CALL: NearGas =
NearGas::new(35_000_000_000_000);
```

when we look at the function **ft_transfer_call**, it construct function data for both **resolve_transfer** and **ft_on_transfer**.

the gas is attached for these call as shown below.
https://github.com/aurora-is-near/aurora-engine/blob/1213f2c7c035aa523601fced8f75bef61b4728ab/engine/src/contract_methods/connector/fungible_token.rs#L244C13-L257

```
let ft_on_transfer_call = PromiseCreateArgs {
    target_account_id: receiver_id,
    method: "ft_on_transfer".to_string(),
    args,
    attached_balance: ZERO_ATTACHED_BALANCE,
    attached_gas: prepaid_gas - GAS_FOR_FT_TRANSFER_CALL - GAS_FOR_RESOLVE_TRANSFER, ---------->>> more gas for
};
let ft_resolve_transfer_call = PromiseCreateArgs {
    target_account_id: current_account_id,
    method: "ft_resolve_transfer".to_string(),
    args: data2,
    attached_balance: ZERO_ATTACHED_BALANCE,
    attached_gas: GAS_FOR_RESOLVE_TRANSFER, ---------------->>> required gas attached.
};
```

Recommendations: Validate the gas available prepaid gas and attach as per necessary. If extra gas needed, attach more some 10% extra.
update the lines of codes as shown below.

https://github.com/aurora-is-near/aurora-engine/blob/1213f2c7c035aa523601fced8f75bef61b4728ab/engine/src/contract_methods/connector/fungible_token.rs#L244C13-L250

```
        let ft_on_transfer_call = PromiseCreateArgs {
            target_account_id: receiver_id,
            method: "ft_on_transfer".to_string(),
            args,
            attached_balance: ZERO_ATTACHED_BALANCE,
            attached_gas: prepaid_gas - GAS_FOR_FT_TRANSFER_CALL - GAS_FOR_RESOLVE_TRANSFER, ------>>> remove
            attached_gas:  GAS_FOR_FT_TRANSFER_CALL, ------>>> add
        };
```

Status: Acknowledged.

Finding: #21

Issue: deposit_event.rs : encoded fee value is not in little endian. this would lead to loss of fee or excess fee when decoding.

Severity: Quality Assurance

Where: eth-connector https://github.com/aurora-is-near/aurora-eth-connector/blob/f73bbe6ac85814a2885a057760843256bc00cd4d/eth-connector/src/deposit_event.rs#L79-L88

Impact: Correct fee value can not be decoded when **parse_on_transfer_message**

Description: In **deposit_event**, when we look at the function **encode** function .

```
#[must_use]
pub fn encode(&self) -> String {
    // The first data section should contain fee data.
    // Pay attention, that for compatibility reasons we used U256 type
    // it means 32 bytes for fee data
    let mut data = U256::from(self.fee.as_u128()).as_byte_slice().to_vec(); -------->> not in little endian
    // Second data section should contain Eth address
    data.extend(self.recipient.as_bytes());
    // Add `:` separator between relayer_id and data message
    [self.relayer.as_ref(), &hex::encode(data)].join(":")
}
```

but when we look at the decoding process in parse_on_transfer_message function. https://github.com/aurora-is-near/aurora-eth-connector/blob/f73bbe6ac85814a2885a057760843256bc00cd4d/eth-connector/src/deposit_event.rs#L59-L64

```
        let mut raw_fee: [u8; 32] = Default::default();
        raw_fee.copy_from_slice(&msg[..32]);
        let fee_u128: u128 = U256::from_little_endian(&raw_fee) ----------->>> in little endian.
            .try_into()
            .map_err(|_| error::ParseOnTransferMessageError::OverflowNumber)?;
        let fee: Fee = fee_u128.into();
```

Recommendations: While encoding the fee in encode function, use little endian based encoding.
this has been done in current running system refer here https://github.com/aurora-is-near/aurora-engine/blob/1213f2c7c035aa523601fced8f75bef61b4728ab/engine/src/contract_methods/connector/deposit_event.rs#L114

Status: Resolved.

Finding: #22

Issue: Storing Private Key in Environment Variables

Severity: Quality Assurance

Where: Makefile#L22

Impact: If an attacker gains access to the environment or the server where these variables are stored, they can easily obtain the private key. This could lead to unauthorized transactions, contract manipulations, or loss of funds. Additionally, if the environment is not properly isolated, other applications or users on the same system might access this sensitive information.

Description: In the migrate command, the private key for signing migration transactions is passed as an environment variable --key **${ACCOUNT_KEY}**. Environment variables are not a secure storage mechanism for sensitive information like private keys. They can be easily accessed by any process running in the same environment, logged inadvertently, or exposed in error messages.

Recommendations: Store private keys in a secure and encrypted storage solution. This could be a hardware security module (HSM), a dedicated secrets management service like AWS Secrets Manager or HashiCorp Vault, or encrypted filesystem storage with strict access controls.

Status: Acknowledged.

08 - Disclaimer

The smart contracts provided to AuditOne 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 ethical nature of the project is not guaranteed by a technical audit of the smart contract. Any owner-controlled functions should be carried out by the responsible owner. Before participating in the project, all investors/users are recommended to conduct due research.

The focus of our assessment was limited to the code parts associated with the items defined in the scope. We draw attention to the fact that due to inherent limitations in any software development process and product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which cannot be free from any errors or failures. These preconditions can impact the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure, which adds further inherent risks as we rely on correctly executing the included third-party technology stack itself. Report readers should also consider that over the life cycle of any software product, changes to the product itself or the environment in which it is operated can have an impact leading to operational behaviors other than initially determined in the business specification.

Contact

auditone.io

@auditone_team

hello@auditone.io

A trust layer of our
multi-stakeholder world.