AUDITONE

Audit Report

AURORA

Omni Bridge

13.01.2025

Table of Contents

CONTENTS

01.
Project Description 3

02.
Project and Audit Information 4

03.
Contracts in scope 5

04.
Executive Summary 6

05.
Severity definitions 7

06.
Audit Overview 8

07.
Audit Findings 9

08.
Disclaimer 30

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

Ubermensch3dot0, Chinepun and Mario Poneder, who are auditors at AuditOne, successfully audited the smart contracts (as indicated below) of Omni Bridge. 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

OmniBridge is a multi-chain asset bridging platform designed to facilitate secure and efficient asset transfers across blockchain networks. Using advanced technologies such as NEAR Multi-Party Computation (MPC) and Chain Signatures, OmniBridge ensures trustless and decentralized cross-chain transactions. The platform bridges the gap between EVM-compatible and non-EVM blockchains, enabling seamless interoperability in a decentralized ecosystem.

This audit focuses on the update of the omni bridge ensuring integrity and security of the contracts.

02-Project and Audit Information

Term	Description
Auditor	Ubermensch3dot0, Chinepun and Mario Poneder
Reviewed by	Luis Buendia and Gracious Igwe
Type	Cross-chain bridging
Language	Solidity & Rust
Ecosystem	Near
Methods	Manual Review
Repository	https://github.com/Near-One/omni-bridge
Commit hash (at audit start)	ffc8cd3156361a9b4d6b79b8d56aa947cff810df
Commit hash (after resolution)	796772cce395d2c72d3031084a914087b10b5941
Documentation	N/A
Unit Testing	N/A
Website	https://aurora.dev/
Submission date	19/11/2024
Finishing date	06/12/2024

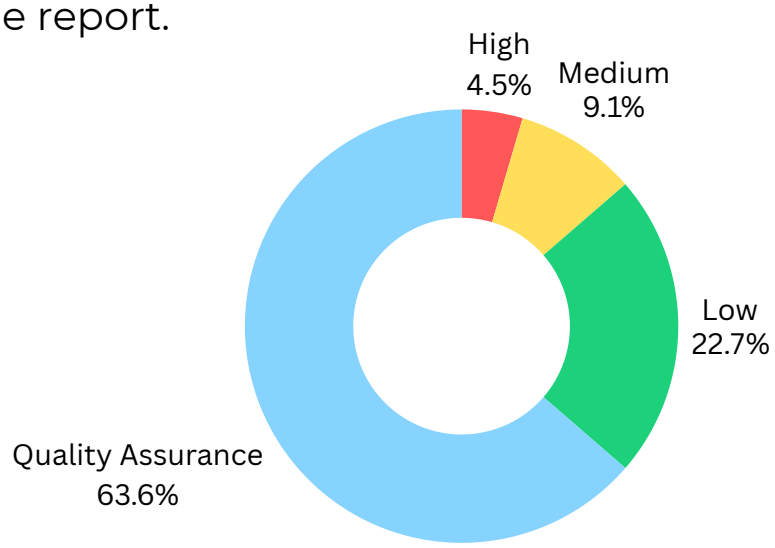
03-Contracts in Scope

Contract in scope:

- contracts/BridgeTokenFactoryWormhole.sol
- contracts/BridgeTokenFactory.sol
- contracts/Borsh.sol
- contracts/BridgeToken.sol
- contracts/SelectivePausableUpgradable.sol
- omni-types/src/lib.rs
- omni-types/src/locker_args.rs
- omni-types/src/evm/events.rs
- omni-types/src/evm/header.rs
- omni-types/src/evm/mod.rs
- omni-types/src/evm/receipt.rs
- omni-types/src/evm/utils.rs
- omni-types/src/mpc_types.rs
- omni-types/src/near_events.rs
- omni-types/src/prover_result.rs
- omni-types/src/prover_args.rs
- nep141-locker/src/lib.rs
- nep141-locker/src/errors.rs
- nep141-locker/src/storage.rs
- omni-prover/evm-prover/src/lib.rs
- wormhole-omni-prover-proxy/src/lib.rs
- wormhole-omni-prover-proxy/src/parsed_vaa.rs
- wormhole-omni-prover-proxy/src/byte_utils.rs
- omni-prover/src/lib.rs

04-Executive summary

Omni Bridge smart contracts were audited between 04-11-2024 and 28-11-2024 by Ubermensch3dot0, Chinepun and Mario Poneder. 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	0	1
Medium	2	2	0
Low	5	4	1
Quality Assurance	14	1	13

05-Severity Definitions

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

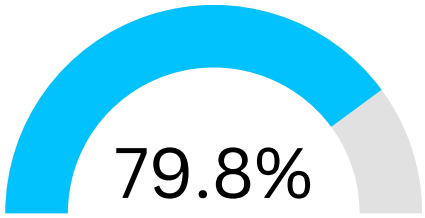
High: Funds or control of the contracts might be compromised directly. Data could be manipulated. We recommend fixing high issues with priority as they can lead to severe losses.

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.

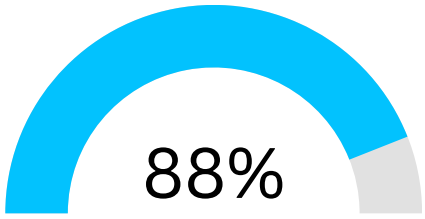
Quality Assurance: Informational and Optimization - Depending on the chain, performance issues can lead to slower execution or higher gas fees. For example, code style, clarity, syntax, versioning, off-chain monitoring (events etc.)

06-Audit Overview



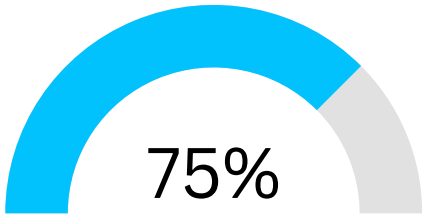
Security score

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: Lack of refund handling in the omni-token contract's mint function can lead to lost tokens.

Severity: High

Where: near/omni-token/src/lib.rs#L109-L115

Impact: In case of failure of the receiver account's ft_on_transfer function, tokens are refunded to the omni-token contract's controller (predecessor account), i.e. the omni-bridge contract. Therefore, the tokens are stuck within the protocol and lost for the user.

Description: When tokens are minted with a msg using the omni-token contract's mint function, there is a subsequent call to ft_transfer_call which eventually delivers the tokens as well as the message to the receiver account_id.

```
fn mint(
        &mut self,
        account_id: AccountId,
        amount: U128,
        msg: Option<String>,
) -> PromiseOrValue<U128> {
        self.assert_controller();

        if let Some(msg) = msg {
                self.token
                        .internal_deposit(&env::predecessor_account_id(), amount.into());

                self.ft_transfer_call(account_id, amount, None, msg)    // <---
        } else {
                self.token.internal_deposit(&account_id, amount.into());
                PromiseOrValue::Value(amount)
        }
}
```

However, according to the NEP 141 standard, if the receiver's **ft_on_transfer** function fails, the transaction is not reverted and the tokens are gracefully refunded to the predecessor account instead, which is the **omni-token** contract's controller, i.e. the **omni-bridge** contract.

Therefore, the tokens are stuck within the protocol, since there seems to be no mechanism to handle this case and recover them from the contract. Furthermore, the tokens are lost from the user's perspective.

Recommendations: It is recommended to check the return value of **ft_transfer_call** and handle to refund case accordingly such that the tokens cannot become stuck and lost from the user's perspective.

Status: Acknowledged.

Finding: #2

Issue: Modifying token decimals via set_token_metadata causes cross-chain desynchronization.

Severity: Medium

Where: lib.rs#L838-L851

Impact: Updating a token's decimal value after it has already been used in a cross-chain context leads to a mismatch between the expected decimals on other chains and the new decimals on NEAR. This discrepancy can cause incorrect asset valuations, user confusion, failed cross-chain transfers, or other unintended outcomes where the bridging logic relies on static or initially defined decimals.

Description: The set_token_metadata function currently allows modifying various token metadata fields, including decimals. If the decimals differ from what the bridging or other chains assume, it breaks the consistency and integrity of the cross-chain communication. Since the bridge logic likely operates under the assumption that decimals are fixed at initialization, changing it at a later stage can lead to incorrect conversions, pricing inaccuracies, and potentially financial loss or arbitrage opportunities.

Recommendations:

- Prevent modification of decimals once the token is initialized. Treat decimals as immutable after the initial setup to avoid cross-chain inconsistencies.
- If dynamic decimals are needed, implement changes on all integrated chains and in the bridging logic to properly handle updates.

Status: Resolved

Finding: #3

Issue: Checked Arithemtic should be used to prevent overflow

Severity: Medium

Where: init_transfer_sol.rs#L49

Impact: An overflow could be reached hence making users send a fraction to the **sol_vault** but it would be seen as if the user paid a huge amount to the **sol_vault.**

Description: This could cause an overflow for extremely high values such that
payload.native_fee + (payload.amount as u64) > u64::MAX

Recommendations: Use **checked_add** over manual addition

Status: Resolved.

Finding: #4

Issue: Restrictive ownership requirement on the **user** signer account limits composability

Severity: Low

Where: init_transfer_sol.rs#L28-L32
init_transfer.rs#L57-L61

Impact: By requiring that the **user** signer account be owned by the system program, the protocol restricts the potential composability of its bridge. Users or other protocols wishing to leverage the bridge with program-derived addresses (PDAs) or smart contract accounts cannot directly interact unless they use a system-owned wallet. This hinders building more complex, automated solutions and seamless integrations with other on-chain programs.

Description: The **user** signer is currently enforced to be system-owned. While this ensures that the user is a standard wallet, it excludes PDAs or accounts from other smart contracts that could act as signers. Such restrictions reduce flexibility, preventing other protocols from programmatically invoking these instructions as signers without resorting to intermediate wallets. In a DeFi ecosystem where composability is key, this limitation may deter integrations and more sophisticated use-cases.

Recommendations: Remove or relax the **owner** constraint on the **user** account. Instead, rely on the Signer constraint to ensure the account is a valid signer.

Status: Acknowledged.

Finding: #5

Issue: Missing **mut** constraint on recipient account in **FinalizeTransferSol.**

Severity: Low

Where: finalize_transfer_sol.rs#L48

Impact: Without marking the **recipient** account as **mut**, the Solana runtime will not treat it as writable. This can lead to runtime errors when attempting to transfer funds to the recipient, as the balance modifications require a writable account.

Description: The **recipient** account should be marked as #[**account(mut)**] to safely allow lamport transfers. Currently, **recipient** is declared as **SystemAccount<'info>** without **mut**. The **transfer** CPI call will attempt to modify the recipient's balance, which requires the account to be writable. Lack of a writable constraint could cause instruction failures at runtime and can potentially break expected functionality.

Recommendations: Add **#[account(mut)]** before the **recipient** field in the **FinalizeTransferSol** accounts struct.

Status: Resolved.

Finding: #6

Issue: ECDSA signature malleability due to non-canonical S-values

Severity: Low

Where: **SignedPayload** struct's **verify_signature** method and surrounding signature verification logic.

Impact: Malleable signatures can allow bad actors to produce multiple valid signatures that represent the same signed message. While nonces in the payloads may mitigate replay attacks, leaving the signature malleability unaddressed can create potential confusion, tooling inconsistencies, or subtle vulnerabilities where logic might depend on a unique canonical signature form.

Description: ECDSA (secp256k1) signatures are susceptible to malleability: the **S** value of the signature can be either "low" or "high," and both still represent a valid signature. Without enforcing a canonical "low-S" form, attackers can alter the signature's **S** value to create multiple distinct but valid signatures for the same message. Although the current design may reduce replay risk by using nonces, it is still a recommended best practice to enforce low-S signatures, ensuring canonicalized signatures and preventing subtle malleability issues.

Recommendations: Enforce the "low-S" rule by checking if **S** is high and revert if it is. For instance, after parsing the signature:

```
if signature.s.is_high() {
    msg!("signature with high-s value");
    return Err(ErrorCode::SignatureVerificationFailed.into());
}
```

Status: Resolved.

Finding: #7

Issue: Unsafe Conversion during token transfer

Severity: Low

Where: init_transfer_sol.rs#49

Impact: Precision loss is possible when a high amount is passed in.

Description: In state/message/init_transfer.rs#L9, the amount is type of u128 but when it is used in init_transfer_sol.rs#49, it does a cast to a u64, this could fail because the initial the integer range of u128 exceeds that of u64.

Recommendations: Either use **u64** for **amount** or make use of the **try_into()** which will fail when the amount value exceeds **u64::MAX**.

Status: Resolved

Finding: #8

Issue: Unnecessary attached deposit with the **omni-token** contract's **mint** function in case of no message.

Severity: Low

Where: near/omni-bridge/src/lib.rs#L911-L918
near/omni-bridge/src/lib.rs#L944-L949

Impact: Overcharging the user for code paths that only involve a token **mint** without a message.

Description: In case of no **msg**, the **mint** function of the **omni-token** contract only calls the underlying **internal_deposit** function which does not require a **ONE_YOCTO** deposit.

```rust
fn mint(
        &mut self,
        account_id: AccountId,
        amount: U128,
        msg: Option<String>,
) -> PromiseOrValue<U128> {
        self.assert_controller();

        if let Some(msg) = msg {
                self.token
                        .internal_deposit(&env::predecessor_account_id(), amount.into());

                self.ft_transfer_call(account_id, amount, None, msg)
        } else {
                self.token.internal_deposit(&account_id, amount.into());    // <---
                PromiseOrValue::Value(amount)
        }
}
```

Therefore, it is not necessary to attach a deposit to the **mint** function when called without a **msg**.

Recommendations: It is recommended to only attach a **ONE_YOCTO** deposit to the **mint** function when a message is involved.

Status: Resolved.

Finding: #9

Issue: Seperate the code into modules

Severity: QA

Where: src/instructions/

Impact: This improves code readability as it seprates permissioned/permisionless instructions.

Description: There are currently 3 priviledges in the program

- admin(calls **initialize** after deployment)
- derived_near_address signature(calls **deploy_token, finalize_transfer** and **finalize_transfer_sol**) with signed payload
- user(can call any of **log_metadata, init_transfer** and **init_transfer_sol**)

However the modules are seperated into 2(admin and user) where user represents all calls that are not done by the admin

Recommendations: Seperate the modules into 3 instead of 2 by adding a module for instructions that are called with **derived_near_address** signature.

Status: Acknowledged.

Finding: #10

Issue: Use anchor constraints ergonomically.

Severity: QA

Where: init_transfer.rs#L59: **owner = wormhole.system_program.key(),**
init_transfer_sol.rs#L30: **owner = wormhole.system_program.key(),**
log_metadata.rs#L36: **constraint = !mint.mint_authority.contains(authority.key),**

Impact: This affects code readability.

Description: The following mentioned code above could be handled better with use of anchor account checks and built in custom error handlers.

Recommendations: For init_transfer_sol.rs#L59 and init_transfer_sol.rs#L30

```
#[account(mut)]
pub user: Signer<'info>,
```

Anchor will check by default that this is a signer and this check is enough as only the signer of this account is allowed to authorise token transfers in this instruction.
Alternatively, the **user** can be removed and the **wormhole.payer** is the account that becomes responsible for token deductions in this instructions

For log_metadata.rs#L36, use anchor attribute to show which custom error is triggered

```
constraint = !mint.mint_authority.contains(authority.key) @ ErrorCode::MintNotAllowed,
```

Status: Acknowledged.

Finding: #11

Issue: Perform padding to state accounts for easier future migrations.

Severity: QA

Where: config.rs

Impact: This could easily help when migrating to a V2 rather than worrying about backwards compatibility.

Description: The **Config** account could be added some extra bytes for padding so it's easier to perform future upgrades than creating new data accounts and ensuring the updated account is backwards compatible.

Recommendations: Add extra bytes to the Config Account

```rust
#[account]
#[derive(InitSpace)]
pub struct Config {
    pub admin: Pubkey,
    pub max_used_nonce: u64,
    pub derived_near_bridge_address: [u8; 64],
    pub bumps: ConfigBumps,
    pub padding: [u8; 100],
}
```

Then add the attached space to the config account upon creation in initialize.rs#L103-L117.

```rust
        self.config.set_inner(Config {
            admin,
            max_used_nonce: 0,
            derived_near_bridge_address,
            bumps: ConfigBumps {
                config: config_bump,
                authority: authority_bump,
                sol_vault: sol_vault_bump,
                wormhole: WormholeBumps {
                    bridge: wormhole_bridge_bump,
                    fee_collector: wormhole_fee_collector_bump,
                    sequence: wormhole_sequence_bump,
                },
            },
            _padding: [0; 100],
        });
```

Status: Acknowledged.

Finding: #12

Issue: Magic numbers/constants should be removed for easier code coverage.

Severity: QA

Where: Length of **derived_near_bridge_address** : config.rs#L23
Length of [**signature**]: mod.rs#L20
Metadata: deploy_token.rs#L35 and log_metadata#L116

Impact: Harder code readability.

Recommendations: Extract all the mentioned magic numbers/constants into the constants.rs file.

Status: Acknowledged.

Finding: #13

Issue: Remove unused accounts or accounts already passed in instructions.

Severity: Quality Assurance

Where: initialize.rs#L88
init_transfer_sol.rs#L15-L19
deploy_token.rs#L46
finalize_transfer_sol.rs#L29 and finalize_transfer_sol.rs#L60
finalize_transfer.rs#L33 and finalize_transfer#L86

Impact: No serious impact other than increasing our transaction size.

Description: Some accounts are passed that are not used in the instruction or have been already passed into the instruction through the **wormholeCPI.**

Recommendations: Delete all the accounts mentioned above.

Status: Acknowledged.

Finding: #14

Issue: Add option for **freeze_authority** when creating wrapped mints.

Severity: Quality Assurance

Where: deploy_token.rs

Impact: Project may want to enable a **freeze_authority** before deploying wrapped tokens on another chain

Description: All wrapped mints created cannot enable a **freeze_authority**.

Recommendations: Add support for **freeze_authority** in the instruction context.

```
#[account(
    init,
    payer = wormhole.payer,
    seeds = [WRAPPED_MINT_SEED, data.payload.token.as_bytes().as_ref()],
    bump,
    mint::decimals = data.payload.decimals,
    mint::authority = authority,
    mint::freeze_authority = freeze_authority,
    mint::token_program = token_program,
)]
pub mint: Box<InterfaceAccount<'info, Mint>>,

pub freeze_authority: Option<SystemAccount<'info>>,
```

Status: Acknowledged.

Finding: #15

Issue: Add support for Token22 in **deploy_tokens**

Severity: Quality Assurance

Where: deploy_token.rs#L31

Impact: Token22 mints is currently not supported for creating wrapped tokens and this is a huge limitation to projects that might want to benefit from the extra extensions that token22 support.

Description: Whenever new wrapped mints are created, it only supports the Token program rather than having flexible support for Token22.
Token22 has more features and is more desirable for projects.

Recommendations: Replace mint and token_program accounts with the following

```
#[account(
    init,
    payer = wormhole.payer,
    seeds = [WRAPPED_MINT_SEED, data.payload.token.as_bytes().as_ref()],
    bump,
    mint::decimals = data.payload.decimals,
    mint::authority = authority,
    mint::token_program = token_program,
)]
pub mint: Box<InterfaceAccount<'info, Mint>>,

pub token_program: Interface<'info, TokenInterface>,
```

Status: Acknowledged.

Finding: #16

Issue: Deploy Token can be called without admin

Severity: Quality Assurance

Where: deploy_token.rs

Impact: If this function is intended to not be called by the **admin**, it means the **admin** is useless and should be removed from the **Config** account so more SOL is not spent on creation of the **Config** Account.

Description: This **deploy_token** creates a new mint account controlled by the **authority** account(Program Account), hence it must be a permissioned instruction.
However, this instruction can only be called with a **SignedPayload** so it is still permissioned but if it is not called by the admin, it gives the admin zero priviledges throughout the program and this is the only instruction that could benefit from an admin account.

Recommendations: The **deploy_token** instruction should only be called by the **admin** or the **admin** should be removed as it is not useful for the program.

Status: Unresolved.

Finding: #17

Issue: An empty payload is posted through wormhole.

Severity: Quality Assurance

Where: initialize.rs#L162

Impact: No severe issue but it would be better to add some details about the transaction to the payload.

Description: The **initialize** instruction is called which creates the config account and then this instruction posts a message to wormhole through CPI.
However, the message is posted without any data in the payload

Recommendations: Add some data in the payload.
Preferably add important data like **admin**, **authority** and **config** into the payload or delete the TODO if no payload is intended to be passed.

Status: Acknowledged.

Finding: #18

Issue: Near Adress could be zero-initialized.

Severity: Quality Assurance

Where: initialize.rs#L95

Impact: While this instruction is only called once, If the caller makes a mistake and passes in an zero-initialized array, It would make the program unusable and a new program will have to be deployed at a different address before the **initialize** instruction can be called.

Description: The **derived_near_address** that will be used for signing payloads is initialized only once and there is no check to ensure that the caller does not make a mistake like passing an empty byte array as the address.

Recommendations: Add a small helper function to ensure the **derived_near_address** is not zero-initialized.

Status: Acknowledged.

Finding: #19

Issue: Missing DAO-controlled setter for the bridge token binary.

Severity: Quality Assurance

Where: near/token-deployer/src/lib.rs#L78-L80

Impact: Changing the **BRIDGE_TOKEN_BINARY** requires a contract upgrade.

Description: The **token-deployer** contract lacks a setter function that allows to update the **BRIDGE_TOKEN_BINARY.** Furthermore, the contract has a DAO role which is currently unused.

Recommendations: It is recommended to implement a setter function for the **BRIDGE_TOKEN_BINARY** which is only accessible by accounts with the DAO role.

Status: Acknowledged.

Finding: #20

Issue: Overrestrictive access control of the **omni-token** contract's **burn** function.

Severity: Quality Assurance

Where: near/omni-token/src/lib.rs#L123

Impact: Overrestrictive access control might impede usability.

Description: The **burn** function of the **omni-token** contract only allows the **controller** to burn its own tokens. However, the function can only burn tokens of the caller (predecessor account) and no one else's. Therefore, the present access control seems overrestrictive.

Recommendations: If token self-burns are accepted, it is recommended to remove the access restriction from the **burn** function.

Status: Acknowledged.

Finding: #21

Issue: The Solana part of the protocol has vulnerable dependencies.

Severity: Quality Assurance

Where: [solana/bridge_token_factory/Cargo.lock]

Impact: Vulnerable dependencies can lead to unexpected attack vectors unrelated to the business logic of the protocol.

Description: The Solana part of the protocol, specifically the **bridge_token_factory** program, has dependencies to Anchor 0.30.1 which in turn has the following vulnerable dependencies:

```
Crate:     curve25519-dalek
Version:   3.2.1
Title:     Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`
Date:      2024-06-18
ID:        RUSTSEC-2024-0344
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0344
Solution:  Upgrade to >=4.1.3

Crate:     ed25519-dalek
Version:   1.0.1
Title:     Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
Date:      2022-06-11
ID:        RUSTSEC-2022-0093
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0093
Solution:  Upgrade to >=2
```

Recommendations: At the time of writing, Anchor 0.30.1 is the most recent version, therefore it is recommended to update the vulnerable sub-dependencies, if required in the context of the protocol.

Status: Acknowledged.

Finding: #22

Issue: Summary of issues known and fixed during audit

Severity: Quality Assurance

Where: 658e7dd
a8b80d2
f03d8da
11ffdb6
07041bd
f102462

Impact: Various.

Description: The following issues were publicly fixed during the audit duration:

- 658e7dd: The user account (signer) of the **init_transfer** and **init_transfer_sol** instructions could be any account that is not necessarily owned by Wormhole. Also, the **recipient** account of the **finalize_transfer_sol** instruction was not required to be mutable although its SOL balance is changed.
- a8b80d2: The **set_token_metadata** function allowed to change the decimals of an already deployed bridge token.
- f03d8da: The **log_metadata** instruction created a **LogMetadataPayload** which includes the trailing **\0** of the **symbol** and **name** strings.
- 11ffdb6: Native fee minting was missing in the **process_fin_transfer_to_near** function. Also, the recipient **ChainKind** was unchecked in the **ft_on_transfer** implementation.
- 07041bd: Insufficient storage deposit accounting in **required_balance_for_account** and **required_balance_for_bind_token** functions. Also, a storage deposit accounting function for bridge token deployment was missing.
- f102462: Missing public accessibility of **token_address_to_id** mapping.

Recommendations: Since the above issues were published and fixed during the audit duration, an explicit finding report or recommendation is not necessary.
However, it is of importance to reassess these fixes during the mitigation review to avoid unintended side effects. Therefore, this issue was created in order to keep track.

Status: Resolved.

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.