coinspect

You build, we defend.

TRICORN

Source Code Review

Soroban Integration

July 2024

Tricorn Bridge
Source Code Review

Version: v240725 Prepared for: Stellar Development Foundation June 2024

Source Code Review

1. Executive Summary

In **April 2024**, **Boosty Labs** engaged Coinspect to conduct a security source code review for the Stellar Soroban integration with the Tricorn Bridge. The objective was to evaluate the security of the Stellar Soroban contracts and their integration into Tricorn's Golang backend.

	Solved	Caution Advised	Resolution Pending
High	3	0	0
Medium	1	4	1
Low	0	1	0
No Risk	3	1	0
Total	**7**	**6**	**1**

Coinspect's analysis identified 3 high-risk vulnerabilities, one of which allows a malicious party to bypass authorization checks and override any Bridge parameter (**TRI-001**). **TRI-002** flags the complete stalling of the bridge and **TRI-003** allows attackers to steal user funds during `bridge_out` calls.

Additionally, the audit identified 6 medium-risk issues: **TRI-004** highlights inefficient layout usage, leading to higher costs. Also, the contract does not account for tokens that have a transfer fee (**TRI-005**). TRI-006 refers to a poor management of the TTL. Additionally, **TRI-007** highlights the lack of adversarial and integration testing. **TRI-008** refers to wrong error handling when processing events in the backend. And finally, **TRI-009** points out the possibility of platform administrators forcing users to pay higher fees.

Lastly, Coinspect identified 1 low-risk concern, **TRI-010**, highlighting an incompatibility within the current bridge implementation and common tokens on EVM-based chains.

2. Summary of Findings

2.1 Findings with pending resolution

These findings indicate potential risks that require some action. They must be addressed with modifications to the codebase or an explicit acceptance as part of the project's known security risks.

Id	Title	Risk
TRI-007	Lack of adversarial unit tests and integration tests	Medium

2.2 Findings where caution is advised

Findings with a risk of `None` pose no threat, but their risk has not been fully mitigated. Any future changes to the codebase should be carefully evaluated to avoid exacerbating these issues or increasing their probability. Document an implicit assumption which must be taken into account. Once acknowledged, these are considered solved.

Id	Title	Risk
TRI-005	Fee on transfer tokens could cause unexpected losses to bridge	Medium
TRI-006	Inconsistent storage TTL handling	Medium
TRI-008	Backend could process duplicate Bridge events	Medium
TRI-009	Platform admin can force users to pay higher fees via front-running	Medium
TRI-010	Unsupported uint256 token value bridge operation	Low

TRI-011	Bridge-in operations do not support high value amounts due to integer overflow	None

2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
TRI-001	Adversaries can modify bridge parameters to steal funds	High
TRI-002	Storage unlimited growth will halt the contract operations	High
TRI-003	Insufficient authorization validation allows adversaries to steal bridge out funds	High
TRI-004	Resource exhaustion due to inefficient storage layout	Medium
TRI-012	Using old Stellar Soroban SDK version	None
TRI-013	Attempting to parse non-existing nonce from BridgeOutEvent event	None
TRI-014	Bridge-in transfer event does not consider the commission in the amount	None

3. Assessment

The scope of this engagement encompasses the Stellar Soroban smart contracts located in the BoostyLabs/tricorn-smart-contracts GitHub repository with the specific commit `2a77de3a5b77a5b0253e43ac1394a9806b995008`. Additionally, it includes the Tricorn bridge backend integration with Stellar, found in the repository BoostyLabs/tricorn with the commit `6b819d74d6e2a4fad5c771770e1ed83b4ebe15a6`. The files specifically included in the scope for the latter are listed below:

```
1084a6595410d373cfac321de49ff5bbd0ab2ee3a701e02eb0469b95eb37ee5f
internal/contracts/stellar/signer/signer.go
56ed79f7322f490c4d22fb2e9c72ac6812b8aceb2420a466fbd8cc92cf97e398
internal/contracts/stellar/txbuilder/txbuilder.go
fdb31110e3ba8710206837d230ce58febd6abec536dc354659c7e262e5d79dd8
internal/eventparsing/stellar/event.go
bb552860b50c6ee76e8942c72fda22d88d979904bd5a2c045de19fd9ef3c7144  connectors/stellar/service.go
86ad9fe6cebbae3d00b08d5c90e323bc1f873b51ff8eb3b20359349063be1f15  connectors/stellar/loop.go
480988667581aa2524b7e86d66406ec8f42578e1f9f4624a8dffc25f1a9b7f9e
connectors/stellar/microservice/microservice.go
aad81c5cd1a3fb8272f61886556d310f20396c982a2c8e2583a2c6b0fe10a1b5  connectors/stellar/connector.go
```

The Tricorn bridge facilitates the transfer of tokens from a source chain to a destination chain and offers four primary functions: `bridge_in`, `bridge_in_burn`, `bridge_out`, and `bridge_out_mint`. The `bridge_in*` functions enable users to initiate transfers from Stellar to any destination chain supported by Tricorn. Conversely, the `bridge_out` functions allow the Tricorn backend to disburse funds that have been transferred from any supported source chain to Stellar. The Stellar Soroban bridge supports both Tricorn-managed tokens (minted/burned) and those not managed by Tricorn. It charges a fee on the bridged amount, capped at 5% and set by default at 1%. These fees are deducted directly from the transfer amount rather than collected separately. This setup gives bridge administrators the possibility to impose higher fees by front-running a `bridge_in*` operation (`TRI-009`).

During a bridge-in operation, the contract moves funds from the user's account to its own balance and then issues a `BridgeInEvent` event detailing the transaction specifics. This event is subsequently captured by the Tricorn backend, which then carries out a bridge-out transaction on the destination chain. The verification of whether the actual transferred balance (amount minus bridge fees) is recorded on the destination chain is beyond the scope of this analysis (refer to `TRI-014`).

3.1 Decentralization

The `bridge_in*` functions should only be initiated by users after they have received a valid signature from the Tricorn bridge backend, which serves as approval for the transaction. Conversely, the `bridge_out*` functions are initiated by a trusted administrative bridge user.

The methodology used to verify the validity of a bridge-in or bridge-out operation was outside the scope of this project. This includes assessing whether a specific bridge operation and its various conditions have been authorized by the bridge. Additionally, it is important to note that the smart contracts do not check if a contract address is on the allow-list. It is assumed by Coinspect that this verification is handled within the backend code that authorizes a `bridge_in` operation.

3.2 Code quality & Testing

The code is straightforward and comprehensible. Tests boasted a 98.04% code coverage, which is optimal for projects of this nature. However, the code did not include unit tests to evaluate adversarial scenarios, such as the absence of authentication for a public function (`TRI-007`). Addressing this could have potentially identified issue `TRI-001` at an earlier stage. Additionally, the code review did not reveal any integration tests, which is vital for projects involving on-chain and off-chain software.

4. Fix Review Assessment

After the fix review, only one issue related to lack of testing (`TRI-007`) remained open. Integration tests are essential for ensuring that the off-chain server interacts correctly with the smart contract. Issues like instance and storage TTL handling could have been detected with a comprehensive integration test suite.

Finally, Coinspect recommends moving all storage-related functions into a separate module, outside the contract implementation. This prevents an adversary from modifying contract-stored variables if any storage management function mistakenly includes a `pub` modifier.

5. Detailed findings

TRI-001

Adversaries can modify bridge parameters to steal funds

Status
Solved

Risk
High

Resolution
Fixed

Impact
High

Likelihood
High

Location

`stellar/contract-bridge/src/lib.rs`

Description

All the administrative functions fail to enforce proper authorization, allowing adversaries to steal deposited commissions or deviate tokens deposits from users by modifying bridge parameters, to name a few.

This is due to the absence of a mechanism to verify whether the platform admin is the actual sender of the transaction. Therefore, an adversary passing the admin address as the parameter can modify the platform signer, the commission collector address or the commission rate. Using the `set_commission_collector` function as an example:

```
pub fn set_commission_collector(
    env: Env,
    address: Address,
    commission_collector: Address,
) -> Result<(), BridgeError> {
    Self::check_is_admin(&env, &address)?;

let mut state: State = Self::get_state(&env)?;

state.commission_collector = commission_collector;

env.storage().instance().set(&STATE, &state);

Ok(())
}
```

The function checks that the parameter sent is the admin within the `check_is_admin` call, but it does not verify if `address` authorized the call.

Recommendation

Use `require_auth()` to ensure that the transaction and its parameters are actually signed by `address`.

Status

Fixed in commit **7d5c4886723be9b2e5f469cb19787706a60c9cd3**. The `set_commission_collector` function now includes a `require_auth` statement to verify that the transaction is signed by the `address`.

TRI-002

Storage unlimited growth will halt the contract operations

Status

Solved

Resolution

Fixed

Risk

High

Impact

High

Likelihood

High

Location

`stellar/contract-bridge/src/lib.rs`

Description

Storage can expand indefinitely, increasing operational costs and potentially stalling the bridge contract.

Nonces are stored in a single map that grows without limits and is re-encoded with each modification. This map is stored in the `instance` storage, which has a capacity limit of 64kb.

```
pub struct State {
    pub default_percent: u128,
    pub default_signer: BytesN<32>,
    pub used_nonces: Map<u32, ()>,
    pub commission_by_token: Map<Address, u128>,
    pub commission_collector: Address,
}
```

```
        env.storage().instance().set(&STATE, &state);
```

This will elevate the operational costs of the bridge until it reaches its storage capacity limit, making it entirely non-functional.

Recommendation

Store the `used_nonces` and `commission_by_token` values in separate variables in persistent storage.

An alternative approach is to have a fixed size array with all the biggest used nonces, and allow no nonce below the minimum. This flexible approach will have a reduced cost while limiting the possibility of using old nonces.

Status

Fixed in commit **9cbe4fddaa6e7ab8b566ddfbd990bf7fc91f017c**. The values from the former `State` struct are now stored under separate `DataKey` in persistent storage. Note however that fields crucial to the contract and not growing over time should remain in instance storage. Since the TTL of instance storage items is tied to the contract TTL, extending the TTL separately would not be required for such items. Refer to the Soroban storage documentation for additional information.

TRI-003

Insufficient authorization validation allows adversaries to steal bridge out funds

Status
Solved

Resolution
Fixed

Risk
High

Impact
High

Likelihood
High

Location

`stellar/contract-bridge/src/lib.rs`

Description

Adversaries can divert bridge out funds to a recipient of their choice by duplicating and front-running a legitimate `bridge_out` or `bridge_out_mint` function call. This is due to the contract only requiring the admin to sign a subset of the functions parameters, allowing an adversary to re-use these signed parameters in another call with a different recipient address.

From the snippet below, note that the `require_auth_for_args` function call only verifies that `address` signed the contract of the token to be bridged and the amount to be bridged.

```
pub fn bridge_out(
    env: Env,
    address: Address,
    token_contract: Address,
```

```
        amount: u128,
        transaction_id: u64,
        source_chain: String,
        source_address: String,
        recipient: Address,
    ) -> Result<(), BridgeError> {
        address.require_auth_for_args((token_contract.clone(),
amount).into_val(&env));

Self::check_is_admin(&env, &address)?;
```

Thus, an adversary monitoring the mempool for `bridge_out` calls to the Tricorn contract could replicate the authorization tree and initiate a new `bridge_out` transaction with an alternate recipient and a higher network fee, effectively front-running the original transaction.

Note this problem is also present in the `bridge_in` functions, although it cannot be directly exploited as the integrity of parameters is protected by a signature. However, should an adversary manage to obtain an arbitrary signature from the backend for a `bridge_in` operation, they could compromise user funds using the technique outlined earlier.

Recommendation

Use the `require_auth` function instead, which expects all the call parameters to be signed by the function caller.

Status

Fixed in commit **7d5c4886723be9b2e5f469cb19787706a60c9cd3**. The `bridge_out`, `bridge_out_mint`, and `bridge_in` functions now enforce authentication on all parameters, ensuring that only the admin and users can call these functions.

TRI-004

Resource exhaustion due to inefficient storage layout

Status
Solved

Resolution
Fixed

Risk
Medium

Impact
Low

Likelihood
High

Location

`stellar/contract-bridge/src/lib.rs`

Description

The storage distribution wastes too many resources on storage read/write due to the inefficient storage design.

The current storage uses one unique state for all values needed to be saved

```
#[contracttype]
#[derive(Clone, Debug, Eq, PartialEq)]
pub struct State {
    pub default_percent: u128,
    pub default_signer: BytesN<32>,
    pub used_nonces: Map<u32, ()>,
    pub commission_by_token: Map<Address, u128>,
    pub commission_collector: Address
```

This causes the entire struct to be read, decoded, modified, re-encoded and saved during operations, instead of only the significant portion.

For example, the `set_stable_comission_percent` function that only needs to modify the `default_percent` entry, reads all the state and re-encodes it when saving.

```
    pub fn set_stable_commission_percent(
        env: Env,
        address: Address,
        stable_commission_percent: u128,
    ) -> Result<(), BridgeError> {
        Self::check_is_admin(&env, &address)?;

 if stable_commission_percent >= MAX_STABLE_COMMISSION_PERCENT {
            return Err(BridgeError::InvalidCommissionPercent);
        }

 let mut state: State = Self::get_state(&env)?;

 state.default_percent = stable_commission_percent;

 env.storage().instance().set(&STATE, &state);

 Ok(())
    }
```

While the `instance` storage is already in memory, the proper decoding and usage of it could be limited to the values relevant to the function.

Recommendation

Save independent values under different `DataKeys`.

Status

Fixed in commit with hash **9cbe4fddaa6e7ab8b566ddfbd990bf7fc91f017c**. The different fields of the `State` struct are now stored separately under different storage keys.

TRI-005

Fee on transfer tokens could cause unexpected losses to bridge

Status
Caution Advised

Resolution
Acknowledged

Risk
Medium

Impact
High

Likelihood
Low

Location

`stellar/contract-bridge/src/lib.rs`

Description

Using fee-on-transfer tokens could cause the platform to release more tokens than received when performing the bridge out operation on the destination chain.

This is due the platform not considering fees charged by fee-on-transfer tokens, and emiting `BridgeInEvent` with the raw `amount` received as a parameter instead of the actual balance received after fees.

```
BridgeInEvent {
    address,
    token_contract,
    amount, <-- THIS IS THE PARAMETER
    gas_commission,
    nonce,
```

```
    transaction_id,
    destination_chain,
    destination_address,
    stable_commission_percent: state.default_percent,
},
```

Recommendation

Use effective received balances in `BridgeInEvent`, instead of fixed parameters. Alternatively, do not use tokens that charge a fee upon transfer.

Status

Acknowledged by the Tricorn bridge team.

TRI-006

Inconsistent storage TTL handling

Status
Caution Advised

Resolution
Partially Fixed

Risk
Medium

Impact
Medium

Likelihood
Medium

Location

`stellar/contract-bridge/src/lib.rs`
`connectors/stellar/service.go`

Description

Coinspect identified several areas for improvement in the management of the contract storage's TTL.

Firstly, the contract lacks storage TTL extension functionality (`extend_ttl` function). This omission necessitates a third party to either periodically extending the TTL or restoring the storage when it becomes archived.

Furthermore, while the in-scope Go code is programmed to restore the storage before sending a transaction if it is archived, it fails to extend the storage TTL. Compounding this issue, all operations involving read-only functions from the backend code do not restore the storage if it has been archived. Consequently, attempts to execute read-only functions, such as `get_commission_collector` through `GetCommissionCollector`, will fail if the storage has been archived.

Lastly, the backend code lacks the capability to determine if a token contract is archived and to restore it if necessary. This deficiency can lead to failures in bridge operations and the `withdraw_commission` function -due to the token contract's storage being archived.

Recommendation

Extend the storage TTL in functions that require sending a transaction/writing the ledger (not read-only).

Ensure that when read-only functions are invoked in the backend, there is a check to confirm that the storage has not been archived, and restore it if necessary.

When invoking functions that interact with token contracts (such as `bridge_*` and `withdraw_commission`), ensure that the token contract storage has not been archived, and restore it if it has.

Status

Partially fixed in commits **7d5c4886723be9b2e5f469cb19787706a60c9cd3** (off-chain server) and **0d5dd70aad1fdf110a0d9d223270dd34ed3461f3** (smart contracts). The changes include:

- Storage TTL is now extended in both write and read calls, and the `callContractMethod` handles restoring any archived data.
- The smart contract now includes an `extend_persistent` function for the backend to periodically extend the storage TTL.

However, the `extendInstanceCall` function called by `GetTokenBalance` only extends the bridge TTL, not the token TTL:

```
	callMethodCallback := func(ctx context.Context)
(*txnbuild.Transaction, *txnbuild.Transaction, error) {
		return service.txBuilder.BuildExtendInstanceTx(ctx,
txbuilder.InvokeHostFunctionData{
			FunctionName:
service.config.ExtendInstanceMethodName,
			ContractAddress:
service.config.BridgeContractAddress,
			SenderAddress:
service.networkID.EncodeAddress(senderAddressResp.PublicKey),
		})
	}
```

Additionally, Coinspect could not find any code responsible for restoring the contract instance (and its storage) if it is archived. Refer to the Stellar Soroban documentation for more details.

TRI-007

Lack of adversarial unit tests and integration tests

Status
Resolution Pending

Resolution
Open

Risk
Medium

Impact
High

Likelihood
Medium

Location

`stellar/contract-bridge/src/test`

Description

Tests, especially automated ones, act as a foundational safety net, ensuring that the source code operates as intended and remains protected from unintended side effects or vulnerabilities.

It is worth noting that multiple issues discovered during this project could have been detected with a proper test suite in place.

The project lacks tests for adversarial scenarios. For example, tests designed to detect unauthorized access to modify contract parameters could have identified issue `TRI-001`.

Automated tests, in particular, act as a critical safety net, ensuring that the code functions as intended and remains protected against unexpected side effects or vulnerabilities.

Recommendation

Add more tests to consider adversarial situations such as unauthenticated or unauthorized actions (eg. modifying the commission collector with an unauthorized address).

Add comprehensive end-to-end integration tests that cover `bridge_in` operations and their required authorization signatures. These tests should also assess the parsing of events generated by these `bridge_in` operations and the subsequent generation of `bridge_out` transactions.

Consider using the `try_...` statement within an assertion instead of declaring the expected panic error. This allows controlling with better precision the line where the error is expected.

Status

Open

TRI-008

Backend could process duplicate Bridge events

Status

Caution Advised

Resolution

Acknowledged

Risk

Medium

Impact

High

Likelihood

Medium

Location

`connectors/stellar/loop.go`

Description

The event retrieval logic may cause the backend to reprocess duplicate events. Reprocessing an event might result in the backend incorrectly releasing more funds than necessary.

The snippet below is from the `ReadRealtimeEvents` function. The `readEvents` function begins by reading and processing events from `currentLedger` or newer, and returns the ledger of the last event read. It then resumes reading and processing from this ledger, creating a risk of reprocessing the most recent events.

```
currentLedger := lastLedgerResp.Sequence -
service.config.EventsReadingGap
        looper :=
thelooper.NewLoop(time.Duration(service.config.EventsReadingIntervalInM
```

```
illiseconds))
        defer looper.Close()

err = looper.Run(context2.JoinContext(ctx, service.gctx), func(ctx
context.Context) error {
                lastLedgerResp, err =
service.sorobanClient.GetLatestLedger(ctx)
                if err != nil {
                        return err
                }

if currentLedger >= lastLedgerResp.Sequence {
                        return nil
                }

previousLedger := currentLedger
                currentLedger, err = service.readEvents(ctx,
currentLedger, eventsChan)
                if err != nil {
                        return err
                }

if previousLedger == currentLedger {
                        currentLedger++
                }

return nil
        })
```

Be aware that any code responsible for de-duplicating processed events falls outside the scope of this engagement and was not reviewed by Coinspect

Recommendation

Ensure that events are fully retrieved on a per-ledger basis, meaning all events for a specific ledger are retrieved atomically. If it's not possible to obtain all events for a given ledger height, discard the current events and attempt retrieval again later.

Implement mechanisms to prevent the reprocessing of duplicate events.

Consider decoupling the logic for reading ledger events from the events processing logic to avoid processing errors impacting the tracking of events retrieved from the chain.

Status

Acknowledged. The Tricorn team mentioned they already have functionality to process received events and filter out already processed ones, but this code is located in another part of their system and was not included in the current engagement's scope.

TRI-009

Platform admin can force users to pay higher fees via front-running

Status

Caution Advised

Resolution

Acknowledged

Risk

Medium

Impact

High

Likelihood

Low

Location

```
stellar/contract-bridge/src/lib.rs
```

Description

The smart contract administrator can force users to pay a higher bridge fee by front-running a `bridge_in` operation.

This problem arises because the `bridge_in` functions do not allow users to specify the maximum fee they are willing to pay, and fees are deducted from the bridged amount instead of being paid separately. Therefore, a malicious admin could front-run a `bridge_in` function by calling `set_stable_commission_percent` and passing a higher fee percentage.

It's important to note that the potential severity of this issue is mitigated by a constraint that the commission cannot exceed or match the `MAX_STABLE_COMMISSION_PERCENT`, which is set at 5% of the funds being bridged.

Recommendation

Allow users to provide the maximum bridge fee they are willing to pay for `bridge_in` operations.

Status

Acknowledged by the Tricorn bridge team.

TRI-010

Unsupported uint256 token value bridge operation

Status

Caution Advised

Resolution

Partially Fixed

Risk

Low

Impact

Medium

Likelihood

Low

Description

Bridge transactions from chains that support `uint256` types, like those EVM-compatible, to Stellar/Soroban might to fail due to incompatible amount types. For instance, the Solidity code for `BridgeInParams` uses a `uint256` amount, while Soroban employs `u128`, which has half the capacity.

Below is the `BridgeInParams` struct, which corresponds to the Solidity code (not covered in this engagement). Note that the `amount` is defined as a `uint256` type.

```
struct BridgeInParams {
    address token;
    uint256 amount;
    uint256 gasCommission;
    string destinationChain;
    string destinationAddress;
    uint256 deadline;
    uint256 nonce;
```

```
    uint256 transactionId;
}
```

On the other hand, the `bridge_out` function only on Soroban allows an `u128` value.

```
pub fn bridge_out(
    env: Env,
    address: Address,
    token_contract: Address,
    amount: u128,
    transaction_id: u64,
    source_chain: String,
    source_address: String,
    recipient: Address,
)
```

The impact of this issue varies based on whether mechanisms are in place on the source chains to unlock or revert a bridge operation if there's an incompatibility, or whether there is a cast from u256 to u128 somewhere in the code. It could result in tokens being locked in the source contract or the loss of gas fees. The likelihood of this occurring is considered low, as bridge administrators can mitigate these incompatibilities by not supporting malicious tokens or those with a high number of decimals.

Recommendation

Do not allow bridge operations that could trigger this incompatibility. Additionally, develop and implement a mechanism to quickly unlock tokens if this issue occurs. Document this limitation and ensure that users are informed.

Status

Partially fixed in commit **2deaad655026859968fa5008e16acc2ce9e25cd0**. The bridge does not provide its signature for bridge-in operations with values higher than `u128`. However, since Soroban tokens use `int128` values, this could still cause an incompatibility. Coinspect recommends limiting bridge operations to `int128` values instead.

TRI-011

Bridge-in operations do not support high value amounts due to integer overflow

Status
Caution Advised

Resolution
Open

Risk
None

Impact
Recommendation

Likelihood
–

Location

`stellar/contract-bridge/src/lib.rs`

Description

An integer overflow in the bridge commission calculation blocks high-value bridge-in operations. This occurs when two unbounded u128 integers are multiplied in the `get_total_commission` function, as illustrated below:

```
amount * self.default_percent / 10_000
```

To generate such overflow, Coinspect prepared a test using an extremely high number (`u128::MAX`). This test triggered a panic, displaying the error message `caught panic 'attempt to multiply with overflow'`.

```
fn test_really_high_commission_overflow() {
    let env = Env::default();
    env.mock_all_auths();
```

```
let contract_id = env.register_contract(None, BridgeContract);
    let client = BridgeContractClient::new(&env, &contract_id);

let sender = Address::generate(&env);
    client.set_admin(&sender);

let signer = generate_keypair();
    let signer_public_key: BytesN<32> = signer_public_key(&env,
&signer);
    client.set_signer(&sender, &signer_public_key);

let gas_commission = 2;
    let max_u128 = u128::MAX;
    let get_total_commission =
        client.get_total_commission(&max_u128, &gas_commission);

println!("get_total_commission: {:#?}", get_total_commission);
}
```

It is important to note that the likelihood of exploiting this issue is low, as it would require an unusually high amount of `bridge_in` tokens, which are only allowed by the platform administrators. Moreover, the impact is minimal because no funds are lost, and the issue can be resolved by conducting smaller bridge transactions.

Recommendation

Consider utilizing a non-overflowing algorithm.

Status

Open. The order of operations still allow a potential overflow:

```
let commission =
amount.checked_mul(default_percent)?.checked_div(10_000)?;
```

Even though checked operations are now used, that does not prevent a `bridge_in` operation from panicking due to overflow. Consider using the `soroban_decimal` library to apply percentages.

TRI-012

Using old Stellar Soroban SDK version

Status
Solved

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
-

Location

```
stellar/Cargo.toml
```

Description

An older dependency is more likely to contain known security issues that have been discovered and exploited over time. Additionally, it can also impact the performance of the contracts as they may lack the optimizations and enhancements that are typically introduced in newer versions, potentially leading to higher fees.

Currently, the project uses the Soroban SDK version `20.0.0`.

Recommendation

Use the latest Soroban SDK version, `20.5.0`.

Status

Fixed in commit with hash **`7d5c4886723be9b2e5f469cb19787706a60c9cd3`**. The project now uses the Soroban SDK version `20.5.0`.

TRI-013

Attempting to parse non-existing nonce from BridgeOutEvent event

Status
Solved

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
-

Location

`internal/eventparsing/stellar/event.go`

Description

The `ParseEvent` function aims to parse a contract's `BridgeOutEvent` event into a `bridgeOutFieldsMap` map, as illustrated below. However, since the `BridgeOutEvent` lacks a `nonce` field, the `nonce` field in `bridgeOutFieldsMap` objects will be `nil`.

Below, a snippet of the `ParseEvent` function is displayed:

```
case BridgeOutEventKey, BridgeOutMintEventKey:
    return event, event.parse(data, bridgeOutFieldsMap)
```

Additionally, the definition of bridgeOutFieldsMap, which contains a `nonce` field:

```
var bridgeOutFieldsMap = map[string]string{
        "token_contract": eventFieldNameTokenContractAddress,
        "amount":         eventFieldNameAmount,
        "nonce":          eventFieldNameNonce,
        "transaction_id": eventFieldNameTransactionID,
        "source_chain":   eventFieldNameChainName,
        "source_address": eventFieldNameChainAddress,
        "recipient":      eventFieldNameUserWalletAddress,
}
```

Finally, the `BridgeOutEvent` event that is emitted by the contract, which does not include a `nonce` value.

```
env.events().publish(
    (&STATE, BRIDGE_OUT_EVENT_KEY),
    BridgeOutEvent {
        address,
        token_contract,
        amount,
        transaction_id,
        source_chain,
        source_address,
        recipient,
    },
);
```

Coinspect advises further evaluation of the impact of this discrepancy since the functionality potentially affected by this mismatch falls outside the scope of this engagement.

Recommendation

Since `bridge_out` functions do not receive any nonce as parameter, consider deleting the the `nonce` value from the `bridgeOutFieldsMap`.

Status

Fixed in commit with hash **9abe856b6cb388d302bfd00c481667946f5c70cf**. The `nonce` field was removed from the `bridgeOutFieldsMap` map.

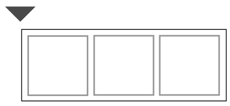
TRI-014

Bridge-in transfer event does not consider the commission in the amount

Status
Solved

Resolution
Acknowledged

Risk
None



Impact
Recommendation

Likelihood
-

Location

`stellar/contract-bridge/src/lib.rs`

Description

The `bridge_in` functions include the entire bridged amount in the `BridgeInEvent` event, without deducting the bridge commission. If not managed correctly, the bridge on the destination chain could release this full amount, leading to operational losses for the bridge by failing to secure its commission.

Below is the `bridge_in` function and the `amount` parameter. Note that this same parameter appears in the `BridgeInEvent` event.

```
pub fn bridge_in(
    ...
    amount: u128,
    ...
) -> Result<(), BridgeError> {
```

```
env.events().publish(
        (&STATE, BRIDGE_IN_EVENT_KEY),
        BridgeInEvent {
            address,
            token_contract,
            amount,
            gas_commission,
            nonce,
            transaction_id,
            destination_chain,
            destination_address,
            stable_commission_percent: state.default_percent,
        },
```

Recommendation

Make sure that the bridge commission is discounted from the `amount` in the destination chain before performing the `bridge_out` operation. Otherwise, discount the bridge commission from the amount before emitting the event.

Status

Acknowledged by the Tricorn bridge team.

6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.