coinspect

You build, we defend.

ICON

Smart Contract Audit

xCall Soroban Implementation

October 2024

xCall

Smart Contract Audit

Version: v241106 Prepared for: Icon Foundation October 2024

Security Assessment

1. Executive Summary

In **September 2024**, **ICON Foundation** engaged Coinspect to perform a smart contract audit of the xCall cross-chain message platform implementation on Stellar Soroban. Also, the project encompassed the review of additional Soroban smart contracts that use the xCall contracts to send cross-chain messages. The objective of the project was to evaluate the security of the different implementations.

The ICON xCall project is a protocol that enables cross-chain communication, allowing blockchains to exchange data and assets.

Solved	Caution Advised	Resolution Pending
High 2	High 1	High 0
Medium 1	Medium 1	Medium 0
Low 2	Low 1	Low 0
No Risk 7	No Risk 1	No Risk 0
Total **12**	Total **4**	Total **0**

The assessment identified three high-risk vulnerabilities that could allow unauthorized draining of the asset manager's token holdings and the triggering of rollbacks without prior authorization. Additionally, Coinspect found that anyone could block updates to source and destination connections in the xCall manager contract.

Coinspect also uncovered two medium-risk issues: one related to insufficient testing, and the other due to a lack of validation when processing a

`MessageRequest` in a reply.

Lastly, the review uncovered three low-risk issues: the lack of privilege segregation in the connection contract, the ability to write data for arbitrary tokens, and the return of empty information rather than an error for non-existent tokens in the Asset Manager contract.

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.2 Finding where caution is advised

Issues with risk in this list have been addressed to some extent but not fully mitigated. Any future changes to the codebase should be carefully evaluated to avoid exacerbating these issues or increasing their probability.

Findings with a risk of None pose no threat, but document an implicit assumption which must be taken into account. Once acknowledged, these are considered solved.

Id	Title	Risk
XCL-001	Anyone can prevent sources and destinations updates on the xCall Manager contract	High
XCL-009	MessageRequest in reply could be sent to wrong destination	Medium
XCL-002	Lack of privilege segregation	Low
XCL-011	Zero value deposits allowed	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

XCL-006	Anyone can trigger rollbacks without authorization	High
XCL-007	Anyone can drain asset manager token holdings	High
XCL-008	Insufficient unit tests and lack of integration tests	Medium
XCL-003	Asset manager contract returns information for non-existing token addresses	Low
XCL-004	Anyone can write token data for arbitrary tokens	Low
XCL-005	Unsafe integer casting	None
XCL-010	Attempting to convert time diff to seconds	None
XCL-012	xCall contract network not computed inside the contract	None
XCL-013	Using the same error for multiple issues hinders testing	None
XCL-014	Unreachable code	None
XCL-015	Deposit function does not enforce destination address	None
XCL-016	Using old Stellar Soroban SDK version	None

3. Scope

The scope was defined to include the following repositories:

- https://github.com/icon-project/xcall-multi at commit `abf40ee78b66f2867ea4d3e964e24fcdc6dbbf4b`. Specifically, the team focused on the xCall core contract and the connection contract.
- https://github.com/balancednetwork/balanced-soroban-contracts at commit `bc20b05d7f94f926dea2c338cc0661e0befa37eb`.

4. Assessment

xCall offers a standard interface for making cross-chain calls between different blockchain networks, consisting of three main components:

- **xCall Core Contracts**: These are the official contracts that dApps interact with to send and receive cross-chain messages.
- **Connections**: Also referred to as sources, destinations, or protocols, these components relay messages between source and destination chains. When sending a message, dApps can choose specific connections or use the default. The xCall contract routes the messages based on these connections.
- **dApps**: The senders and receivers of cross-chain messages. The contracts fitting this category reviewed during this project are the **Asset Manager**, **Balanced Dollar**, and **xCall Manager**.

As outlined in xCall's documentation, the security of the platform depends on the integrity of its underlying connections. It is the dApp's responsibility to validate the source connection/protocol during the `handleCallMessage` process. Any address can send messages to xCall, and they are assumed valid by default, but the dApp can discard them if protocols are found to be invalid. Given this, ICON must address all relevant security considerations to help dApps developers to avoid accepting malicious or spoofed messages. Additionally, providing a dApp contract template could ensure dApp owners can leverage existing security features.

xCall users face two costs when sending messages: xCall platform fees and a per-connection fee, which is determined by the connection administrator. Receivers only need to pay the chain's transaction fee to execute the message.

It's important to note that these fees are uncapped, allowing administrators to raise them without limit at any time. However, Soroban's token transfer authorization scheme prevents front-running attacks to increase fees, as users must explicitly approve and sign the fee transfer. If fees increase, the transaction simply fails. Nonetheless, Coinspect recommends setting a hard cap on these fees to prevent any automatic system from automatically paying extremely high fees.

Finally, due to the time constraints of this engagement and the absence of integration tests, Coinspect was unable to perform dynamic testing on the various components involved in the message exchange process. As a result, Coinspect recommends an additional review focused on these interactions and their error-handling mechanisms.

4.1 Security assumptions

Coinspect consultants conducted the assessment based on the following assumptions:

- The ICON Foundation operates with good intent and does not engage in malicious behavior.
- Administrative control of the contracts is governed by a multisig setup.
- dApps trust the connections (protocols), as malicious connections can spoof `from` addresses and replay messages. The software components used by connection operators to relay messages operate correctly.
- Off-chain code, which was not part of this assessment, manages the extension of contract instance storage. Some reviewed contracts handle instance and persistent storage extension in their public-facing functions, while others do not.

4.2 Decentralization

The project exhibits a significant level of centralization for several reasons:

- All contracts are upgradeable.
- The xCall team has the ability to modify the core contract code, adjust protocol fees, and change the default connection.
- The default centralized connection contract is intended to be controlled by ICON.

Additionally, it should be noted that connections face no penalties for failing to deliver messages or for delivering incorrect information. Due to the protocol's design, if a message relies on multiple connections, it can be halted if any one of the connections fails to deliver it.

4.3 Testing

As highlighted in `XCL-008`, in addition to the absence of an integration test suite, Coinspect observed that the current unit tests are insufficient, as they fail to cover essential functionality of the core contracts. Furthermore, the unit testing suite would benefit from more adversarial tests.

The code coverage for each contract in scope is as follows:

- xCall: 77.62% coverage, 711/916 lines covered
- centralized-connection: 80.92% coverage, 123/152 lines covered

- asset_manager: 61.36% coverage, 405/660 lines covered
- balanced_doller: 53.35% coverage, 342/641 lines covered
- xcall_manager: 52.38% coverage, 275/525 lines covered

Coinspect strongly recommends enhancing the overall testing across the project.

4.4 Code quality

Overall, aside from the informational issue regarding unreachable code (XCL-014), Coinspect found the code generally clear and easy to follow. However, there is potential for improving its quality. Adding documentation to each core function to briefly explain the functionality, inputs, and outputs in a NatSpec-like format would be beneficial.

Furthermore, the technical documentation could be enhanced by offering more detailed explanations of the flow for a message request, execution, reply and rollback process.

5. Detailed Findings

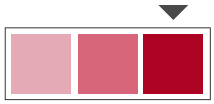
XCL-001

Anyone can prevent sources and destinations updates on the xCall Manager contract

Status
Caution Advised

Resolution
Partially Fixed

Risk
High



Impact
High

Likelihood
High

Location

`balanced-soroban-contracts/contracts/xcall_manager/src/contract.rs:68`

Description

Anyone can cause a Denial-of-Service (DoS) to the contract by filling up its instance storage. This could prevent ICON governance from adding new source and destination protocols. Additionally, an adversary could block the `handle_call_message` function from executing correctly by front-running each

call with a `remove_action` call, which would remove the white-listed action, causing `handle_call_message` to throw a `NotWhiteListed` error.

This issue arises due to two vulnerabilities:

1. Lack of authorization enforcement on the `white_list_actions` function.
2. White-listed actions being stored in the contract's instance storage, which is limited to 64kb (resource limits reference). Once the storage limit is reached, no additional data can be stored unless existing entries are removed.

Below, note that the `white_list_actions` function can be called by anyone:

```
pub fn white_list_actions(e: Env, action: Bytes) {
    let actions = WhiteListActions::new(DataKey::WhiteListedActions);
    actions.add(&e, action);
}
```

And the `add` function, which stores the `value` in instance storage:

```
impl WhiteListActions {
    pub fn new(key: DataKey) -> Self {
        Self { key }
    }

pub fn add(&self, env: &Env, value: Bytes) {
        let mut list = self.get(env);
        list.push_back(value);
        env.storage().instance().set(&self.key, &list);
    }
}
```

Recommendation

Implement authorization checks on the `white_list_actions` function.

Store allow-listed actions in persistent storage instead. Consider storing sources and destinations and any other information of variable length in persistent storage as well.

Status

Partially fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The `white_list_actions` function now enforces access control; however, there remains a risk of DoS due to instance storage exhaustion if an excessive number of actions are stored.

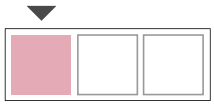
XCL-002

Lack of privilege segregation

Status
Caution Advised

Resolution
Acknowledged

Risk
Low



Impact
Medium

Likelihood
Low

Location

`xcall-multi/contracts/soroban/contracts/centralized-connection/src/contract.rs:87`

Description

The connection contract currently defines two distinct roles or addresses authorized to interact with it: the upgrade authority, responsible for upgrading the contract's functionality, and the admin, who is permitted to set and collect fees as well as to support the message exchange operations. In the event of the admin's credentials, which would likely be stored in a server, being compromised, adversaries could not only disrupt the connection's operations but also steal the accumulated fees.

Here's an example of the `recv_message` function, which enforces the admin's authorization:

```
    pub fn recv_message(
        env: Env,
        src_network: String,
        conn_sn: u128,
```

```
        msg: Bytes,
    ) -> Result<(), ContractError> {
        helpers::ensure_admin(&env)?;
```

Recommendation

Assign role responsibilities considering the type of operations and whether the private keys need to be stored in an off-chain software component.

Status

Acknowledged. The development team indicated that the Xcall specification assigns the same privilege for centralized connections across all other chains, with plans to consider adjustments in the future.

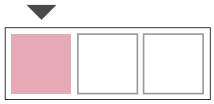
XCL-003

Asset manager contract returns information for non-existing token addresses

Status
Solved

Resolution
Fixed

Risk
Low



Impact
Low

Likelihood
Low

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:96`

Description

The asset manager contract returns an empty, yet valid, `TokenData` object when a query is made for a non-existing token address. This behavior could mislead consumers of this information into believing that the token address is stored in the contract and taking actions based on that assumption.

The issue arises because the `read_token_data` function returns a default `TokenData` object if the `token_address` is not found in the contract. This function is invoked by the publicly accessible `get_rate_limit` function.

```
pub fn read_token_data(env: &Env, token_address: Address) -> TokenData
{
    let default = TokenData { percentage: 0, period: 0, last_update: 0,
```

```
current_limit: 0 };
    let key = DataKey::TokenData(token_address);
    env.storage().persistent().get(&key).unwrap_or(default)
}
```

Recommendation

Instead of returning an empty response, throw an error when attempting to retrieve data for a non-existing token.

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The `read_token_data` function is no longer public, and the `get_rate_limit` function now checks if the token exists. Note however `read_token_data` still returns empty information.

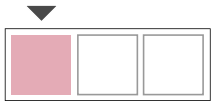
XCL-004

Anyone can write token data for arbitrary tokens

Status
Solved

Resolution
Fixed

Risk
Low



Impact
Low

Likelihood
Low

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:123`

Description

Anyone can cause the contract to update the `last_update` field for an arbitrary token address due to the absence of authorization enforcement in the public `verify_withdraw` function, shown below. Furthermore, since there is no check to verify whether the token was previously stored, combined with the issue highlighted in `XCL-003`, this can also be done for non-existing tokens.

```
pub fn verify_withdraw(env: Env, token: Address, amount: u128) ->
Result<bool, ContractError> {
    let balance = Self::get_token_balance(&env, token.clone());
    let limit = Self::calculate_limit(&env, balance, token.clone())?;
    if balance - amount < limit {
        panic_with_error!(&env, ContractError::ExceedsWithdrawLimit);
    };
    let mut data: TokenData = read_token_data(&env, token.clone());
    data.current_limit = limit as u64;
```

```
    data.last_update = env.ledger().timestamp();
    write_token_data(&env, token, data);
    Ok(true)
}
```

Note that for non-existing tokens, the `limit` property would still default to zero.

A similar situation occurs with the `reset_limit` function, which it only stores the token address key.

```
pub fn reset_limit(env: Env, token: Address) {
    let balance = Self::get_token_balance(&env, token.clone());
    let mut data: TokenData = read_token_data(&env, token.clone());
    data.current_limit = (balance * data.percentage as u128 / POINTS)
as u64;
    write_token_data(&env, token, data);
}
```

Recommendation

Ensure that the `token` address exists in storage before proceeding. Additionally, consider implementing an access control mechanism unless the current design intentionally allows unrestricted access.

Status

Fixed in commit **a07bcd707fe9ae6b55332a12e1d6049c89aad4d5**. The `read_token_data` function no longer returns empty data for non-existing tokens. Additionally, the `reset_limit` function now enforces authentication from the admin.

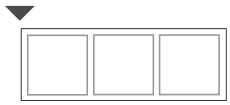
XCL-005

Unsafe integer casting

Status
Solved

Resolution
Fixed

Risk
None



Impact
Recommendation

Likelihood
–

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:322`

`balanced-soroban-contracts/contracts/balanced_doller/src/balanced_dollar.rs:34`

Description

The `transfer_token_to` function casts the u128 `amount` variable to i128 without considering potential overflows. If the u128 value is greater than the maximum value of i128, the cast will result in a wrapped value, resulting in a negative number.

```
fn transfer_token_to(e: &Env, from: Address, token: Address, to:
Address, amount: u128) { //ok
    let token_client = token::Client::new(e, &token);
    token_client.transfer(&from, &to, &(amount as i128));
}
```

A similar situation occurs in the `_cross_transfer` function from the Balanced Dollar contract:

```
_burn(&e, from.clone(), amount as i128);
```

Note however that the standard token implementation in Soroban does not allow negative values, and therefore this issue is deemed as info. Keep in mind that using non-standard token implementations allowing negative transfers would allow exploitation of this problem.

Recommendation

Safely handle `amount` values higher than the maximum value of i128.

Review and fix the rest of the implementations where this problem was introduced.

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The `transfer_token_to` and `_cross_transfer` functions now enforce that the `amount` does not exceed `i128::MAX`.

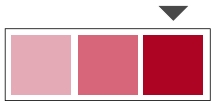
XCL-006

Anyone can trigger rollbacks without authorization

Status
Solved

Resolution
Fixed

Risk
High



Impact
High

Likelihood
High

Location

`xcall-multi/contracts/soroban/contracts/xcall/src/handle_message.rs:161`

Description

Anyone can force a `ResponseFailure` for a any `CallMessageWithRollback`, which allows executing the rollback. The impact depends on the `dapp` handling the rollback. For instance, a bridge `dapp` would allow an adversary to redeem the deposited funds on both chains. This is enabled by the lack of authentication enforcement on the `handle_error` function, which allows an adversary to pass any arbitrary source address to meet the rollback's protocols requirements.

Once the rollback is enabled, they can also execute it since the `execute_rollback` function does not require authentication.

Anyone can execute the public `handle_error` function while providing a valid source address, which is not authenticated.

```
pub fn handle_error(env: Env, sender: Address, sequence_no: u128) ->
Result<(), ContractError> {
    handle_message::handle_error(&env, sender, sequence_no)
}
```

Which executes the `handle_error` function from the `handle_message` crate.

```
pub fn handle_error(env: &Env, sender: Address, sequence_no: u128) ->
Result<(), ContractError> {
    let cs_message_result = CSMessageResult::new(
        sequence_no,
        CSResponseType::CSResponseFailure,
        Bytes::new(&env),
    );
    handle_result(&env, &sender, cs_message_result.encode(&env))
}
```

Note that this does not prevent valid sources from reporting successful responses. However, off-chain systems processing a response failure without looking for further results could be tricked into processing a fake response.

Recommendation

Enforce authentication on the `handle_error` function.

Status

Fixed on commit **e76f07b60a739b8d1e19d052865acc86f38601c2**. The `handle_error` function now enforces authentication of the `sender` parameter.

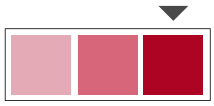
XCL-007

Anyone can drain asset manager token holdings

Status
Solved

Resolution
Fixed

Risk
High



Impact
High

Likelihood
High

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs 303:319`

Description

Anyone can withdraw any token held by the contract to an arbitrary address. This is due to the `withdraw` function displayed below accessible by anyone, which allows providing the contract's address as the `from` parameter to

```
pub fn withdraw(
    e: &Env,
    from: Address,
    token: Address,
    to: Address,
    amount: u128,
) -> Result<(), ContractError> {
    if amount <= 0 {
        return Err(ContractError::AmountIsLessThanMinimumAmount);
    }
```

```
let verified = Self::verify_withdraw(e.clone(), token.clone(),
amount)?;
    if verified {
        Self::transfer_token_to(e, from, token, to, amount);
    }
    Ok(())
}
```

Recommendation

Remove the `pub` modifier from the function declaration.

Additionally, consider relocating all non-public functions to a separate module outside of the contract. This will ensure that internal functions are not accidentally exposed to the public.

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The `withdraw` function is no longer public.

PoC (Proof-of-Concept)

Coinspect confirmed this vulnerability through the test below, which allows tokens to be transferred from the contract to an arbitrary address via the `withdraw` function, without triggering any `AuthorizedInvocation`.

```
#[test]
fn test_withdraw() {
    let ctx = TestContext::default();
    let client = AssetManagerClient::new(&ctx.env, &ctx.registry);
    ctx.init_context(&client);

client.configure_rate_limit(&ctx.token, &300, &300);

let bnusd_amount = 100000u128;
    let token_client = token::Client::new(&ctx.env, &ctx.token);
    let stellar_asset_client: token::StellarAssetClient =
        token::StellarAssetClient::new(&ctx.env, &ctx.token);
    stellar_asset_client.mint(&client.address, &((bnusd_amount * 2) as
i128));

let to_address = Address::generate(&ctx.env);
```

```
ctx.env.mock_all_auths();
    client.withdraw(&client.address, &ctx.token, &to_address,
&bnusd_amount);
    // std::println!("AUTHORIZATION {:?} ----", ctx.env.auths()); -->
This is empty

assert_eq!(token_client.balance(&to_address), bnusd_amount as i128);

assert_eq!(
        ctx.env.auths(),
        std::vec![
            // NO AUTHS REQUIRED
        ]
    );

}
```

To execute it locally, place this snippet in the `asset_manager_test.rs` file and execute the following command:

```
tests::asset_manager_test::test_withdraw -- --nocapture
```

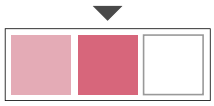
XCL-008

Insufficient unit tests and lack of integration tests

Status
Solved

Resolution
Fixed

Risk
Medium



Impact
Medium

Likelihood
Medium

Description

Coinspect did not identify an integration testing suite covering all contracts within scope. Since proper source validation and authorization are critical for this platform, the absence of integration tests may obscure issues stemming from the interaction between these components.

Automated tests, in particular, serve as a crucial safeguard, ensuring that the source code functions as expected and is shielded from unintended side effects or vulnerabilities.

The xCall contract's code coverage is currently at 77%, which may overlook important functionality. For example, Coinspect identified that there are no unit tests for the `execute_call` and `handle_error` functions, which could have exposed issue `XCL-006`.

Recommendation

Add unit tests for the `handle_error` and `execute_call` functions in the xCall implementation. Improve test code coverage for the xCall contract. Add adversarial tests to account for scenarios such as handling a request from multiple sources with an unexpected message for a given sequence number, or attempting to execute data that does not match the stored SHA-256 hash of the request.

Implement integration tests to cover the interaction between the different components supporting end-to-end message passing. Include adversarial integration tests to simulate potential attack scenarios.

Status

Fixed. The development team added greatly improved the xCall unit testing suite, found on commit **de9a2931d0a772413e2c4b1a4bf95abdc7b66207**. Additionally, they provided integration tests at https://github.com/bcsainju/balanced-stellar-deploy, although these do not cover malicious case scenarios. On the other hand, they decided not to write additional test for the balanced contracts.

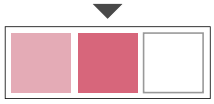
XCL-009

MessageRequest in reply could be sent to wrong destination

Status
Caution Advised

Resolution
Acknowledged

Risk
Medium



Impact
Medium

Likelihood
Medium

Description

The `handle_reply` function processes a `MessageRequest` contained within a `CSMessageResult`. However, it does not validate whether the reply's destination (`to`) matches the address that initiated the original request. This omission allows the `MessageRequest` to be sent to a different destination.

In the code snippet provided, the `handle_reply` function verifies that the source of the reply (`reply.from`) matches the original destination address, but it does not ensure the reply's destination is correct.

```
pub fn handle_reply(
    env: &Env,
    rollback: &Rollback,
    reply: &mut CSMessageRequest,
) -> Result<(), ContractError> {
    if rollback.to().nid(&env) != reply.from().nid(&env) {
        return Err(ContractError::InvalidReplyReceived);
    }
    let req_id = storage::increment_last_request_id(&env);
```

```
    event::call_message(
        &env,
        reply.from().to_string(),
        reply.to().clone(),
        reply.sequence_no(),
        req_id,
        reply.data().clone(),
    );

    reply.hash_data(&env);
    reply.set_protocols(rollback.protocols.clone());
    storage::store_proxy_request(&env, req_id, &reply);

    Ok(())
}
```

Since the Soroban implementation under review does not insert such requests in a result during the `execute_message` call, Coinspect could not identify a clear method of exploiting this issue.

Recommendation

Evaluate whether this behavior is allowed on the platform. If not, ensure that the reply's destination address is validated to match the one that sent the initial `MessageRequest`.

Status

Acknowledged. The development team states that this behavior is allowed and replies can be sent to other destination contracts.

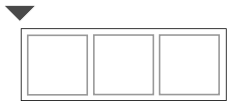
XCL-010

Attempting to convert time diff to seconds

Status
Solved

Resolution
Fixed

Risk
None



Impact
Recommendation

Likelihood
–

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:152`

Description

When computing the `time_diff` in the `calculate_limit` function, the resulting difference of two timestamps is divided by `1000` in an attempt to convert it from miliseconds to seconds. Note however that timestamps in Soroban are already expressed in seconds.

```
let time_diff = (&env.ledger().timestamp() - last_update) / 1000;
```

This problem was not detected during the testing phase as the `test_configure_rate_limit_panic` test uses the `get_withdraw_limit` and `verify_withdraw` outputs to verify this calculation, both of which use the `calculate_limit` function.

Recommendation

Express the `time_diff` in seconds instead.

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The time diff is now expressed in seconds by removing the division from the formula.

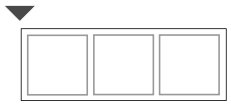
XCL-011

Zero value deposits allowed

Status
Caution Advised

Resolution
Open

Risk
None



Impact
Recommendation

Likelihood
–

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:192`

Description

The `send_deposit_message` function in the asset manager contract allows zero-value deposits, as there are no restrictions in place to prevent this.

While this is not a security vulnerability, it does present inconsistent behavior when compared to cross-chain withdrawals, which do not permit zero-value withdrawals. The snippet below was obtained from the same contract.

```
pub fn withdraw(
    e: &Env,
    from: Address,
    token: Address,
    to: Address,
    amount: u128,
) -> Result<(), ContractError> {
    if amount <= 0 {
```

```
            return Err(ContractError::AmountIsLessThanMinimumAmount);
        }
```

Recommendation

Consider adding a restriction to prevent zero-value transfers.

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The `deposit` function, which calls `send_deposit_message` ensures that the deposited amount is higher than zero.

XCL-012

xCall contract network not computed inside the contract

Status
Solved

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
–

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:282`

Description

Currently, the `xcall_network_address` value (the `NetworkAddress` for the xCall contract) is set via the `config` parameter passed to the `initialize` function. Since this value is derived from the xCall contract address (`config.xcall`) and the network ID, including it as a separate parameter introduces an unnecessary risk. An unauthorized modification of this value will enable spoofing of the `from` address in the `handle_call_message` function.

The xCall contract's `NetworkAddress` is included in the `initialize` function as part of the `config`:

```
pub fn initialize(env: Env, registry: Address, admin: Address, config:
ConfigData) {
```

Which defines this parameter as `xcall_network_address`:

```
pub struct ConfigData {
    ...
    pub xcall_network_address: String,
    ...
}
```

Recommendation

Derive the `xcall_network_address` directly from the `xcall` address and the network ID.

Status

Fixed in commit **f81ebcf16bda5d6c6bc3a39856ff10ffed5b8074**. The xCall network address is now obtained from the xCall contract via the `get_network_address` function.

XCL-013

Using the same error for multiple issues hinders testing

Status
Solved

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
–

Location

`xcall-multi/contracts/soroban/contracts/xcall/src/handle_message.rs:44`

Description

The `handle_request` function returns the same error for two distinct issues. As a result, if one of these validations fails, tests may obscure the actual cause, as they cannot differentiate which validation triggered the error.

```
    if src_net != from_net {
        return Err(ContractError::ProtocolsMismatch);
    }
    let source = sender.to_string();
    let source_valid = is_valid_source(&env, &source, src_net,
&req.protocols())?;
    if !source_valid {
        return Err(ContractError::ProtocolsMismatch);
    }
```

Recommendation

Consider using different errors for different causes.

Status

Fixed on commit **de9a2931d0a772413e2c4b1a4bf95abdc7b66207**. The code now throws the `NetworkIdMismatch` error when the networks differ.

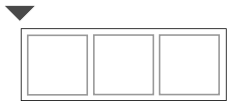
XCL-014

Unreachable code

Status
Solved

Resolution
Fixed

Risk
None



Impact
Recommendation

Likelihood
–

Location

`contracts/xcall_manager/src/contract.rs:152`

Description

Unreachable code makes smart contracts harder to understand and maintain. In the example below, the instructions in the first `if` block prevent the second `if` block from executing.

```
if !Self::verify_protocols(e.clone(), protocols.clone())? {
    return Err(ContractError::ProtocolMismatch);
};

let method = ConfigureProtocols::get_method(&e.clone(), data.clone());

let sources = read_sources(&e);
if !Self::verify_protocols_unordered(protocols.clone(),
sources).unwrap() {
    if method != String::from_str(&e.clone(), CONFIGURE_PROTOCOLS_NAME)
{
        return Err(ContractError::UnknownMessageType);
    }
```

```
    Self::verify_protocol_recovery(&e, protocols)?;
}
```

The `verify_protocols` function runs the same logic and receives the same parameters as the second `if` block, making the latter unreachable.

```
pub fn verify_protocols(e: Env, protocols: Vec<String>) -> Result<bool,
ContractError> {
    let sources: Vec<String> = read_sources(&e);

let verified = Self::verify_protocols_unordered(protocols, sources)?;
    return Ok(verified);
}
```

Recommendation

Remove the portion of code that is unreachable.

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca**. The `verify_protocols` function is no longer executed.

XCL-015

Deposit function does not enforce destination address

Status
Solved

Resolution
Acknowledged

Risk
None

Impact
Recommendation

Likelihood
–

Location

`balanced-soroban-contracts/contracts/asset_manager/src/contract.rs:170`

Description

The `deposit` function in the asset manager contract allows a None `to` parameter, raising uncertainty about whether this behavior is intentional and, if so, the rationale for permitting transfers to an empty destination.

As demonstrated in the code below, the `deposit` function assigns an empty string to the `to` parameter if it is None:

```
pub fn deposit(
    e: Env,
    from: Address,
    token: Address,
    amount: u128,
    to: Option<String>,
    data: Option<Bytes>,
```

```
) -> Result<(), ContractError> {
    let deposit_to = to.unwrap_or(String::from_str(&e, ""));
```

Recommendation

Ensure the `to` parameter is neither None nor empty. Otherwise, document and communicate this behavior clearly.

Status

Acknowledged. The development team stated that if a destination address is not provided, ICON Balanced treats the sender's address as the destination address.

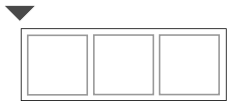
XCL-016

Using old Stellar Soroban SDK version

Status
Solved

Resolution
Fixed

Risk
None



Impact
Recommendation

Likelihood
–

Location

`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 `20.5.0`.

Recommendation

Use the latest Soroban SDK version, `21.7.3`

Status

Fixed on commit **72f86192d51baf53af86d1eb7a76637059d838ca** from the Balanced contracts (SDK version `21.6.0`) and commit **de9a2931d0a772413e2c4b1a4bf95abdc7b66207** from xCall multi contracts (SDK version `21.7.4`).

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.