Canto
Findings & Analysis Report

2023-09-29

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Canto system written in Go. The audit took place between June 20—June 23 2023.

Wardens

24 Wardens contributed reports to the Canto:

1. sces60107
2. dontonka
3. 3docSec
4. Franfran
5. Team_FliBit (14si2o_Flint and Naubit)
6. Lirios
7. yaarduck
8. hihen
9. Rolezn
10. erebus
11. seerether
12. 0xNightRaven
13. 0xSmartContract
14. Udsen
15. kutugu
16. squeaky_cactus
17. DevABDee
18. solsaver
19. vuquang23
20. max10afternoon
21. nadin
22. Shogoki
23. kaveyjoe

This audit was judged by 0xean.

Final report assembled by PaperParachute and liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 2 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 1 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 12 reports detailing issues with a risk rating of LOW severity or non-critical.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Canto repository, and is composed of 4 files written in the Go programming language and includes 289 lines of code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (1)

[H-01] Pre-defined limit is different from the spec

Submitted by sces60107*, also found by* Franfran*,* Team_FliBit*,* dontonka*,* Lirios*, and* 3docSec

https://github.com/code-423n4/2023-06-canto/blob/main/Canto/x/coinswap/keeper/swap.go#L212
https://github.com/code-423n4/2023-06-canto/blob/main/Canto/x/coinswap/types/params.go#L34

Impact

In the spec, the pre-defined limit of ETH is 0.01 ETHs. But the actual limit in the code is not 0.01 ETH which could result in misleading.

Proof of Concept

In the spec, it said that the pre-defined limit of ETH is 0.01 ETHs.
https://github.com/code-423n4/2023-06-canto/blob/main/README.md#swap

> For risk management purposes, a swap will fail if the input coin amount exceeds a pre-defined limit (10 USDC, 10 USDT, 0.01 ETH) or if the swap amount limit is not defined.

But in `x/coinswap/types/params.go` , the actual limit of ETH is 1*10e17 which is 0.1 ETH.

```
// Parameter store keys
var (
        KeyFee                    = []byte("Fee")
        KeyPoolCreationFee        = []byte("PoolCreationFee")
        KeyTaxRate                = []byte("TaxRate")
        KeyStandardDenom          = []byte("StandardDenom")
        KeyMaxStandardCoinPerPool = []byte("MaxStandardCoinPerPo
        KeyMaxSwapAmount          = []byte("MaxSwapAmount")

        DefaultFee                    = sdk.NewDecWithPrec(0, 0)
        DefaultPoolCreationFee        = sdk.NewInt64Coin(sdk.Def
        DefaultTaxRate                = sdk.NewDecWithPrec(0, 0)
        DefaultMaxStandardCoinPerPool = sdk.NewIntWithDecimal(10
        DefaultMaxSwapAmount          = sdk.NewCoins(
                sdk.NewCoin(UsdcIBCDenom, sdk.NewIntWithDecimal(
                sdk.NewCoin(UsdtIBCDenom, sdk.NewIntWithDecimal(
```

```
                sdk.NewCoin(EthIBCDenom, sdk.NewIntWithDecimal(1
        )
)
```

The limit is used in `swap.GetMaximumSwapAmount`. Wrong could harm the risk management.
https://github.com/code-423n4/2023-06-canto/blob/main/Canto/x/coinswap/keeper/swap.go#L212

```
func (k Keeper) GetMaximumSwapAmount(ctx sdk.Context, denom stri
        params := k.GetParams(ctx)
        for _, coin := range params.MaxSwapAmount {
                if coin.Denom == denom {
                        return coin, nil
                }
        }
        return sdk.Coin{}, sdkerrors.Wrap(types.ErrInvalidDenom,
}
```

Recommended Mitigation Steps

0.01 ETH should be `sdk.NewIntWithDecimal(1, 16)`.

Assessed type

Error

0xean (judge) increased severity to High

tkkwon1998 (Canto) confirmed and commented on duplicate issue 8:

> Agreed, this issue is valid as limit is 10x higher than it should be. Although losses are still minimal (0.1 eth at most), I agree with high risk since funds can be lost if pools are manipulated.

Medium Risk Findings (1)

[M-01] Potential risk of using `swappedAmount` in case of swap error

Submitted by yaarduck, *also found by* hihen, yaarduck, erebus, sces60107, Rolezn, *and* seerether

https://github.com/code-423n4/2023-06-canto/blob/main/Canto/x/onboarding/keeper/ibc_callbacks.go#L93-L96
https://github.com/code-423n4/2023-06-canto/blob/a4ff2fd2e67e77e36528fad99f9d88149a5e8532/Canto/x/onboarding/keeper/ibc_callbacks.go#L124

Impact

In case the swap operation failed, the module should continue as is with the erc20 conversion and finish the IBC transfer. This is the relevant part of the code that swallows the error:

```
swappedAmount, err = k.coinswapKeeper.TradeInputForExactOutput(c
if err != nil {
    logger.Error("failed to swap coins", "error", err)
}
```

Notice that in case of an error, `swappedAmount` will still be written to.

Later on in the code, it is used to calculate the conversion amount:

```
convertCoin := sdk.NewCoin(transferredCoin.Denom, transferredCoi
```

The `swappedAmount` is trusted to have a zero value in this case. While this is currently true in the existing code, variables returned in error states should not be trusted and should be overwritten.

Currently all error states return `sdk.ZeroInt()` unless the swap was executed correctly, but it might change in a future PR.

Proof of Concept

1. Run this patch, it will cause TradeInputForExactOutput to always error with a swappedAmount > 0 .

```
diff --git a/Canto/x/coinswap/keeper/swap.go b/Canto/x/coins
index 67e04ef..a331bcf 100644
--- a/Canto/x/coinswap/keeper/swap.go
+++ b/Canto/x/coinswap/keeper/swap.go
@@ -2,6 +2,7 @@ package keeper

 import (
        "fmt"
+
        sdk "github.com/cosmos/cosmos-sdk/types"
        sdkerrors "github.com/cosmos/cosmos-sdk/types/errors

@@ -160,51 +161,8 @@ Buy exact amount of a token by specifyi
 @param receipt : address of the receiver
 @return : actual amount of the token to be paid
 */
-func (k Keeper) TradeInputForExactOutput(ctx sdk.Context, i
-       soldTokenAmt, err := k.calculateWithExactOutput(ctx,
-       if err != nil {
-               return sdk.ZeroInt(), err
-       }
-
-       // assert that the calculated amount is less than th
-       // max amount the buyer is willing to pay.
-       if soldTokenAmt.GT(input.Coin.Amount) {
-               return sdk.ZeroInt(), sdkerrors.Wrap(types.E
-       }
-       soldToken := sdk.NewCoin(input.Coin.Denom, soldToker
-
-       inputAddress, err := sdk.AccAddressFromBech32(input.
-       if err != nil {
-               return sdk.ZeroInt(), err
-       }
-       outputAddress, err := sdk.AccAddressFromBech32(outpu
-       if err != nil {
-               return sdk.ZeroInt(), err
-       }
-
-       standardDenom := k.GetStandardDenom(ctx)
-       var quoteCoinToSwap sdk.Coin
-
-       if soldToken.Denom != standardDenom {
```

```
-                quoteCoinToSwap = soldToken
-        } else {
-                quoteCoinToSwap = output.Coin
-        }
-
-        maxSwapAmount, err := k.GetMaximumSwapAmount(ctx, qu
-        if err != nil {
-                return sdk.ZeroInt(), err
-        }
-
-        if quoteCoinToSwap.Amount.GT(maxSwapAmount.Amount) {
-                return sdk.ZeroInt(), sdkerrors.Wrap(types.E
-        }
-
-        if err := k.swapCoins(ctx, inputAddress, outputAddre
-                return sdk.ZeroInt(), err
-        }
-
-        return soldTokenAmt, nil
+func (k Keeper) TradeInputForExactOutput(_ sdk.Context, _ t
+        return sdk.NewIntFromUint64(10000), fmt.Errorf("swap
 }
```

2. Add this test to `x/onboarding/keeper/ibc_callbacks_test.go` :

```
{
    "swap fails with swappedAmount / convert remaining ibc t
    func() {
        transferAmount = sdk.NewIntWithDecimal(25, 6)
        transfer := transfertypes.NewFungibleTokenPacketData
        bz := transfertypes.ModuleCdc.MustMarshalJSON(&trans
        packet = channeltypes.NewPacket(bz, 100, transfertyp
    },
    true,
    sdk.NewCoins(sdk.NewCoin("acanto", sdk.NewIntWithDecimal
    sdk.NewCoin("acanto", sdk.NewIntWithDecimal(3, 18)),
    sdk.NewCoin(uusdcIbcdenom, sdk.NewIntFromUint64(10000)),
    sdk.NewInt(24990000),
},
```

The test will still fail because of another unimportant check, but the important check will pass - the address will have `sent-swappedAmount` vouchers converted, and the

rest will be kept.

It means swappedAmount was used even though the swap function failed.

Tools Used

IDE.

Recommended Mitigation Steps

Zero the `swappedAmount` variable in the error case:

```
swappedAmount = sdk.ZeroInt()
```

Assessed type

Other

tkkwon1998 (Canto) confirmed and commented on duplicate issue 80:

> The warden is correct in that a swap could go only half-completed if the function is used incorrectly, but all of the checks should occur before it ever gets to the swap function to ensure it succeeds fully. In our case, all of the checks are done in `TradeInputForExactOutput`. If the warden believes we are missing any checks which may cause the swap to only half-complete, then that would be great.

> I will mark this as valid because, as the warden mentioned, if any future upgrades use the swap function, there may be a potential that they do not do the necessary checks.

Low Risk and Non-Critical Issues

For this audit, 9 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **squeaky_cactus** received the top score from the judge.

The following wardens also submitted reports: solsaver, vuquang23, DevABDee, nadin, Rolezn, Shogoki, kaveyjoe, *and* 3docSec.

Issue Summary

2 low issues and 8 non-critical found.

Id	Issue
L-01	Misleading method GoDoc `calculateWithExactInput`
L-02	Misleading method GoDoc `GetPoolBalances`
N-01	Dead GoDoc params for `TradeExactInputForOutput`
N-02	Dead GoDoc params for `TradeInputForExactOutput`
N-03	Params has both value and reference receivers
N-04	Deprecated method on `EmitEvent` used
N-05	Silent fail on no whitelisting
N-06	Silent fail on recipient account being a module
N-07	Silent fail on missing ERC20 mapping
N-08	Silent fail on trading pair disable

[L-01] Misleading method GoDoc `calculateWithExactInput`

GoDoc paramater of `soldTokenDenom` on `swap:calculateWithExactInput` is not a function parameter, however there is a boughtTokenDenom parameter (which would be an inversion of the GoDoc description).

Recommended Mitigation

```
x/coinswap/keeper/swap.go:L33

- @param soldTokenDenom : received token's denom
+ @param boughtTokenDenom : demonination of token brought
```

[L-02] Misleading method GoDoc `GetPoolBalances`

GoDoc method comment for `pool:GetPoolBalances` does not describe the method correctly (it may be a copy, paste & edit error from the proceeding method in the file).

Recommended Mitigation

```
x/coinswap/keeper/pool.go:L69

- // GetPoolBalances return the liquidity pool by the specified
+ // GetPoolBalances returns the coin balances of the liquidity
```

[N-01] Dead GoDoc params for `TradeExactInputForOutput`

The GoDoc for `swap:TradeExactInputForOutput` contain two parameters are not in the method signature and there are no likely matching candidates, these lines should be deleted as they'll only serve to sow confusion.

swap.go L68-69
@param sender: address of the sender
@param receipt: address of the receiver

Recommended Mitigation

```
x/coinswap/keeper/swap.go:L68-69

-@param sender: address of the sender
-@param receipt: address of the receiver
```

[N-02] Dead GoDoc params for `TradeInputForExactOutput`

The GoDoc for `swap:TradeInputForExactOutput` contain two parameters are not in the method signature and there are no likely matching candidates, these lines should be deleted as they'll only serve to sow confusion.

swap.go L68-69
@param sender: address of the sender

@param receipt: address of the receiver

Recommended Mitigation

```
x/coinswap/keeper/swap.go:L159-160

-@param sender : address of the sender
-@param receipt : address of the receiver
```

[N-03] Params has both value and reference receivers

Params has two methods, one uses pointer and the other a value reciver. GoLang documentation recommends choosing either one or the other.
(Arguments being consistency, readability and performance)
https://go.dev/doc/faq#methods_on_values_or_pointers

The Validate method is currently using a value receiever, this can be changed to a pointer without side affect (avoiding copying the struct on each call).

Recommended Mitigation

Change the `Params` from value to reference.

```
x/onboarding/keeper/params.go:L89

- func (p Params) Validate() error {
+ func (p *Params) Validate() error {
```

[N-04] Deprecated method on `EmitEvent` used

EmitEvent is deprecated in favor of EmitTypedEvent.
https://pkg.go.dev/github.com/cosmos/cosmos-sdk/types@v0.45.9#EventManager.EmitEvent

```
x/onboarding/kepper/ibc_callbacks.go:L97-105
                        ctx.EventManager().EmitEvent(
                                sdk.NewEvent(
```

```
                coinswaptypes.EventTypeS
                sdk.NewAttribute(coinswa
                sdk.NewAttribute(coinswa
                sdk.NewAttribute(coinswa
                sdk.NewAttribute(coinswa
                sdk.NewAttribute(coinswa
            ),
```

Recommended Mitigation

Use `EmitTypedEvent`.

[N-05] Silent fail on no whitelisting

There is no logging on the early return condition of no whitelisted channels found for the destination channel.

Logging this return condition would provide a data point for observability, supporting system monitoring and issue diagnosis.

Recommended Mitigation

```
x/onboarding/kepper/ibc_callbacks.go:L56
+logger.Error("no whitelist channel found for %s",packet.Destina
```

[N-06] Silent fail on recipient account being a module

There is no logging on the early return condition when the recipient account is a module.

Logging this return condition would provide a data point for observability, supporting system monitoring and issue diagnosis.

Recommended Mitigation

```
x/onboarding/kepper/ibc_callbacks.go:L71
+logger.Error("recipient account is a module %s",account.GetAddr
```

[N-07] Silent fail on missing ERC20 mapping

There is no logging on the early return condition when no ERC20 mapping is found for the denom.

Logging this return condition would provide a data point for observability, supporting system monitoring and issue diagnosis.

Recommended Mitigation

```
x/onboarding/kepper/ibc_callbacks.go:L120
+logger.Error("no ERC20 mapping found for %s",transferredCoin.De
```

[N-08] Silent fail on trading pair disable

There is no logging on the early return condition when the trading pair is disabled.

Logging this return condition would provide a data point for observability, supporting system monitoring and issue diagnosis.

Recommended Mitigation

```
x/onboarding/kepper/ibc_callbacks.go:L127
+logger.Error("disabled trading pair with id %s",pairID)
```

tkkwon1998 (Canto) confirmed and commented:

> Really nice QA report.

Audit Analysis

For this audit, 3 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such topics as architecture, mechanism, or approach. The **report highlighted below** by **0xNightRaven** received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, Udsen, and kutugu.

Approach to Auditing the Canto Platform

Phase 1: Documentation and Video Review

- Start with a comprehensive review of all documentation related to the Canto platform, including the whitepaper, API documentation, developer guides, user guides, and any other available resources.
- Watch any available walkthrough videos to gain a holistic understanding of the system. Pay close attention to any details about the platform's architecture, operation, and possible edge cases.
- Note down any potential areas of concern or unclear aspects for further investigation.

Phase 2: Manual Code Inspection

- Once familiarized with the operation of the platform, start the process of manual code inspection.
- Review the codebase section by section, starting with the core smart contract logic. Pay particular attention to areas related to slippage handling, non-existing pools, and frontrunning prevention.
- Look for common vulnerabilities such as reentrancy, integer overflow/underflow, improper error handling, etc., while also ensuring the code conforms to best programming practices.
- Document any concerns or potential issues identified during this stage.

Phase 3: Review of IBC Documentation and External Integration

- Thoroughly read the Inter-Blockchain Communication (IBC) protocol documentation. Ensure that Canto's implementation is in line with the standard, and note any deviations.
- Evaluate the handling of cross-chain transactions and the robustness of Canto's solution against possible external attacks or vulnerabilities.
- Examine how the platform integrates with external systems (e.g., oracles, other blockchains). Investigate how the platform handles communication errors or

discrepancies in data from these systems.

- Note any potential weak points in the IBC and external integration implementation.

Architecture Recommendations

Onboarding Module is well-designed with a proper sequence of operations. However, a few aspects should be noted:

1. Error Handling: The error handling in the onboarding middleware should be enhanced. Currently, even if the swap or conversion fails, it does not revert IBC transfer. The onboarding process is non-atomic, meaning that the asset transferred to the Canto network will still remain in the Canto network. A robust error handling system can greatly improve the system's reliability.
2. Middleware Ordering: The middleware ordering seems to be well thought out. However, the impact of the order on the performance and reliability of the transactions should be assessed in a real-time scenario.
3. Whitelisting Channels: The system works only when transactions are performed through a whitelisted channel. While this reduces risk, it could limit the flexibility of the system. Thus, further analysis of the trade-off between security and flexibility would be beneficial.

Codebase Quality Analysis

The quality of the codebase appears to be of high quality and well-structured. A key aspect is the separation of concerns with different modules handling different aspects of the system, like onboarding, IBC transfers, and the Coinswap module.

The usage of Go modules also increases the maintainability and readability of the code. The code is well-commented, which can be beneficial for future developers.

However, it is recommended to use linters and static code analyzers to identify and rectify any potential security vulnerabilities, bugs, or code smells.

Centralization Risks

There are risks related to the centralization of control in the system. A key risk area is the whitelisted channels for auto swap and convert. If the governance of the

whitelisted channels is not properly decentralized, it may lead to centralization.

Moreover, the parameters like `enable_onboarding` and `auto_swap_threshold` could be manipulated by a centralized entity. A clear governance structure should be implemented to mitigate these risks.

Mechanism Review

The system utilizes the Inter-Blockchain Communication (IBC) protocol for the transfer of assets and the onboarding module is used to convert these assets into Canto tokens and ERC20 tokens automatically.

The system uses an Automated Market Maker (AMM) model for swapping the assets. The Coinswap module handles the AMM functions. The overall mechanism seems to be well thought out.

However, the system should analyze the potential risks and vulnerabilities associated with the AMM model, such as impermanent loss and the risk of liquidity pools being drained.

Handling of Slippage

Slippage is a significant issue in decentralized trading platforms, which occurs when the price changes during a transaction due to the liquidity's volatility.

From the audit, it is observed that the Canto smart contract does not have explicit handling for slippage. This could lead to users experiencing unfavorable trade execution, especially for large orders that may move the price significantly.

Recommendations

1. Implement a slippage tolerance feature, where a user can specify a maximum acceptable price slippage for their trade. If the actual price slippage exceeds this tolerance, the transaction should be reverted.
2. To further mitigate slippage, consider using price oracles to provide accurate and timely pricing data, which can help in better trade execution.

Handling of Non-Existing Pools

Currently, it appears that the system does not have specific handling for non-existing pools. This could potentially lead to failed transactions and unsatisfactory user experiences.

Recommendations

1. Implement a mechanism to check the existence of a pool before initiating a transaction. If a pool does not exist, the transaction should be prevented or reverted, and the user should be notified appropriately.
2. Consider a feature that allows users or liquidity providers to create new pools when necessary, fostering a more flexible and user-friendly ecosystem.

Systemic Risks

A systemic risk in this system is the dependence on the Gravity Bridge and Cosmos SDK. Any vulnerability or bug in the Gravity Bridge or Cosmos SDK would directly impact the Canto platform.

The system also heavily relies on the proper functioning of the IBC. Any issues in the IBC protocol could potentially cause serious implications for the system.

Conclusion

The Canto platform, with its onboarding module, provides a user-friendly approach to convert assets into Canto and ERC20 tokens. However, the system should pay attention to error handling, potential centralization risks, and dependencies on external systems.

Time spent:

20 hours

tkkwon1998 (Canto) confirmed

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top