WUλFMAN CORPORATION

Audit Token to Token

January 2022

Contents

Disclaimer

This report does not provide any warranty or guarantee regarding the absolute bug-freenature of the technology analyzed.

This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Wulfman Corporations position is that each company and individual are responsible for their own due diligence and continuous security. Wulfman Corporations goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Introduction

This audit was commanded to Wulfman Corporation, in quality of main contributor and expert of LigoLANG, by SmartChain

The object of the audit is the analysis of the Token to Tokenin order to identify vulnerabilities and contract optimizations in the source code.

The contract targets the Tezos blockchain and was developed in LigoLANG. The auditing methods consist in manual review

The auditing process paid special attention to ensuring that the contract logic is coherent and implements the specification and the best testing schemes.

Overview

Project summary

Project Name	Token to Token
Publisher	SmartChain
Platform	Tezos
Language	LigoLANG(cameligo flavor)
Codebase	https://github.com/Smartlinkhub/token-to-token/tree/master
Original commit	65f2ef1d8c816262b70ab782d99cfc5a98408f06
Contract adress	
Contract url	

Audit summary

Auditer	Wulfman Corporation
Delivery date	January 2022
Scope	
Methodology	Manual review
Tezos version	
Tezos client version	
LigoLANGversion	0.31.0

Vulnerability summary

Total issues	3
Critical	0
Major	0
Medium	1
Minor	2
Informational	0

Code Quality summary

Total improvements	10
Maintenance	4
Scalability	1
Readability	3
Origination cost	0
Gas cost	2

Vulnerability

Contents

V1. Incorect check for "balance of" callback

Category	Severity	Location	Status
Potential inaccuracy	Minor	dex_token2token.mligo	Fixed

Description

The `fa2_balance_callback` extract the amount from a list return by calling the `balance_of` entrypoint of an *FA2* contract. This extracting is valid when the list as a size bigger than 1, which shouldn't be posible since this entrypoint is always call with a list of 1 element.

Solution

the matching should differenciate between `[(_,amnt)]` (valid) and `_` (invalide), instead of `[]` and `(_, amt)::_xs`

N.B : this is not very important since the contract use other mecanism for checking this.

V2. Get dex address doesn't work as expected

Category	Severity	Location	Status
Fonctionality not working	Medium	factory.mligo	Fixed

Description

I didn't find a specification for the view `get_dex_address`. Naively, the expected behavior should be, I provide an unorder pair (A, B) and the view return the address of the liquidity pool for the pair A/B, which is the same as B/A. The current behavior of this view is to take an order pair A/B, reorder it based on an input parameter (which is not needed, the caller could have send the pair in the right order) and look for this order pair. There is two potential issues with this design :

1. if the A/B dex is deploy but the caller ask for pair B/A the `get_dex_address` will fail with error `DEX_ADDRESS_NOT_FOUND_IN_FACTORY`, when the dex is in factory and you won't be able to lauch it. This will make the complete chain of call fails and requires a complete retry with swapping the parameters

2. the `direction` field is redondant with the already order A/B and add extra layer of processing for nothing (more of a code quality issues)

Solution

For *2.*, you can ask the user to always feed A/B in the order of `direction=false`, overall it should change much for the client in term of code complexity and gas usage (probably a slight decrease) but it will remove code complexity on the server size

For *1.*, there is several possibilitis with different tradeoff

1. You can keep the current specification which keep the code simple but for an external contract (C) to use the views (V), the contract caller needs to first parse the list of deployed contract by looking up the tezos data on internet before calling C and V
2. Change the behavior of the views to return an error instead of an exception, which would allow the (C) to handlae and error in the use of (V)
3. You can search for both A/B and B/A which remove the limitation but increase the computation hence the gas cost
4. You can store both A/B and B/A in the map pointing to the same address. Which also remove the limitation but incease the storage size

V3. Guaranty the symetry by desing

Category	Severity	Location	Status
Potential inaccuracy	Minor	dex_token2token.mligo	Fixed

Description

There is unavoidable complexity in `dex_token2token` due to the dex being symetric with regard to the pair A/B but the pair in michelson are ordered and the `(from_,to_)` pair is an order A/B or B/A pair.

The way it is handle in the contract is with an boolean `a_to_b` and several local check to do the same processing on (A, B) or (B, A) leading to code deduplication. This duplication may lead to copy error or desynchronisation during development.

example in `swap` :

```
    if a_to_b
    then
      (((storage.token_pool_a + tokens_sold) - feeA_SMAK),
       ((storage.token_pool_b - bought) - feeB_SMAK))
    else
      (((storage.token_pool_a - bought) - feeA_SMAK),
       ((storage.token_pool_b + tokens_sold) - feeB_SMAK)) in
```

and

```
      if a_to_b
      then
        ((token_a_transfer storage Tezos.sender Tezos.self_address tokens_sold),
         (token_b_transfer storage Tezos.self_address t2t_to bought))
      else
        ((token_a_transfer storage Tezos.self_address t2t_to bought),
         (token_b_transfer storage Tezos.sender Tezos.self_address
            tokens_sold)) in
```

Solution

Use a layer of abstraction, using `let from_,to_ = A,B` or `let from_,to_ = B,A` based on `a_to_b`. Do all the processing on `from_` and `to_` and then use the value of `a_to_b` to project the result back in storage like this

---

```
let swap (param : token_to_token) (storage : storage) =
    let { to = t2t_to; token_sold; min_tokens_bought; a_to_b; deadline } = param in
    let () = check_self_is_not_updating_token_pool storage in
    let () = check_deadline deadline in
    (* abstraction *)
    let
    ↪  (token_pool_from,token_from_id,token_from_address),(token_pool_to,token_to_id,token_to_address)
    ↪  =
        let A,B =
            (storage.token_pool_a,storage_token_id_a,storage.token_address_a),
            (storage.token_pool_b,storage_token_id_b,storage.token_address_b)
    if a_to_b then(A,B) else (B,A)
    in
    let (bought, feeFrom_SMAK, feeTo_SMAK) =
        match aorb_is_smak with
        | Some aorb_is_smak ->
            compute_out_amount_when_A_or_B_is_SMAK a_to_b aorb_is_smak tokens_sold
            ↪  token_pool_from token_pool_to
        | None -> compute_out_amount tokens_sold token_pool_from token_pool_to, 0n, 0n
    in
    if bought < min_tokens_bought
    then
    (failwith
    error_TOKENS_BOUGHT_MUST_BE_GREATER_THAN_OR_EQUAL_TO_MIN_TOKENS_BOUGHT : (operation list
    ↪  * storage))
    else (
    let (new_pool_from, new_pool_to) =
      ((token_pool_from + tokens_sold) - feeFrom_SMAK),
      ((token_pool_to - bought) - feeTo_SMAK)
    in
    let new_pool_from : nat =
        match is_nat new_pool_from with
```

```
29          | None ->
30              (failwith error_TOKEN_POOL_MINUS_TOKENS_BOUGHT_IS_NEGATIVE : nat)
31          | Some difference -> difference in
32      let new_pool_to =
33      match is_nat new_pool_to with
34      | None ->
35         (failwith error_TOKEN_POOL_MINUS_TOKENS_BOUGHT_IS_NEGATIVE : nat)
36      | Some difference -> difference in
37      let (op_token_from_transfer, op_token_to_transfer) =
38          ((token_transfer token_from_address token_from_id Tezos.sender Tezos.self_address
            ↪  tokens_sold),
39          (token_transfer token_to_address   token_to_id   Tezos.self_address t2t_to bought))
40      let op_token_transfer =
41          opt_operation_concat op_token_from_transfer op_token_to_transfer in
42      (* projection *)
43      let ((new_pool_a,feeA_SMAK),(new_pool_b,feeB_SMAK)) =
44          let a,b = ((new_pool_from,feeFrom_SMAK),(new_pool_to,feeTo_SMAK)) in
45          if a_to_b then (a,b) else (b,a)
46      in
47      let new_history =
48          Big_map.update "token_pool_a" (Some new_pool_a) storage.history in
49      let new_history =
50          Big_map.update "token_pool_b" (Some new_pool_b) new_history in
51      let amounts_and_fees_out =
52          compute_fees storage new_pool_a new_pool_b feeA_SMAK feeB_SMAK in
53      let ops_pay_fees =
54          withdraw_or_burn_fees storage amounts_and_fees_out.reserve_fee_in_A
55          amounts_and_fees_out.reserve_fee_in_B in
56      let storage =
57      {
58          storage with
59          token_pool_a = amounts_and_fees_out.amount_in_A;
60          token_pool_b = amounts_and_fees_out.amount_in_B;
61          history = new_history;
62          last_k =
63          (amounts_and_fees_out.amount_in_A * amounts_and_fees_out.amount_in_B)
64      } in
65     ((operation_concat op_token_transfer ops_pay_fees), storage))
```

Code Quality

Contents

Q1. Use of preprocessor guard

Category	Impact	Location	Status
Informational	Maintenance	all ligo file	Acknowledge

Description

The codebase use the good c-style practice of file guard to avoid infinite loop in include cycles. Note that ligo contrary to C doesn't separate signature from code and while the guard prevent the preprocessor from looping indefinetly. It we fails in the typechecker with an error that may be not so clear.

Solution

To avoid dependency cycle, prefer using the `#import` pragma which detect dependency cycle and print the correcponding error. N.B : Since the code is produce from OCaml, you should already have dune checking for depenency cycle.

Q2. Including file in the middle of the code

Category	Impact	Location	Status
Bad design	Maintenance	factory.mligo	Acknowledge

Description

The design of `#include` in ligo, allow to do put the content of a file anywhere in the code. the file `factory.mligo` take adventage of this design by injecting the code of the contract directly inside a `michelson_insertion`. This is convenant to keep the file in sync with the contract. But it disable

ligo typechecking for the contract being deploy. Furthermore, in the futur LigoLANGwon't allow `#include` to be used anywhere in the code.

Solution

Instead of deploy through a `michelson_insertion`,
You can directly use the ligo command `Tezos.create_contract` For this, you need to first import the ligo contract :

```
#import "dex_token2token.mligo" "Token2Token"

let deploy_dex (init_storage : dex_storage) : (operation * address) =
    Tezos.create_contract Token2Token.main (None : key_hash option) Tezos.amount
    ↪  init_storage
```

(replace `main` by the name of the entrypoint if different)

Q3. Separate checking

Category	Impact	Location	Status
Code smell	Readability	dex_misc.mligo	Fixed

Description

The function `update_token_pool_internal_checks` group several test that are not related. the entypoint that call the checks is less readible, i.e. it takes more effort to go look in another file to see the list of assertion that are being check by the entrypoint.

Solution

Separate the different checking and put them in the entrypoint.

Q4. Unecessary checking for own entrypoint

Category	Impact	Location	Status
Informational	Gas cost	dex_token2token.mligo	Acknowledge

Description

The function `update_token_pool_aux` calls `get_entrypoint2` which call a `Tezos.get_entrypoint_opt` on itself and return an error if the entrypoint doesn't exist. By contruction, we know if the entrypoint is present in the contract that we are developping. If it is a good practice to check for this during development to spot mistakes. In deployement, this will lead to extra instruction for checking a condition that is always true. This is unecessary cost for the user.

Solution

Replace `Tezos.get_entrypoint_opt` with `Tezos.get_entrypoint`.
Alternatively, by `Option.unopt (Tezos.get_entrypoint_opt ...)`.

Q5. Putting SMAK on the same side of the pair

Category	Impact	Location	Status
Suggestion	Maintenance & Gas cost	factory & dex	Acknowledge

Description

Since A/B pool is the same as B/A pool, you could had the convention that a pair with smak will have SMAK always on A or B. This would require a bit of extra logic in factory at the creation of the pool but it would simplify the logic of the dex contract and possibly some gas cost.

Note : This is not compatible with the proposal for V3.

Q6. Misplacement of code chunk

Category	Impact	Location	Status
Code smell	Maintenance & Readability	contract	Acknowledge

Description

Some of the code is misplaced. For instance, the datastructure `dex_storage` is located in `comman_types` instead of `dex_types`

Q7. Miscelanous performance improvement

Category	Impact	Location	Status
Suggestion	Gas cost	DEX	Acknowledge

Description

The entrypoint use a nested variant inside the main variant. As variant are compiled to a tree of `some`, this design produce an inbalance tree that result in more operation to manipulate it in average. The same happens when using `[@layout:comb]`, and it should be avoided when not necessary. It doesn't seem that `add_liquidity` and `remove_liquidity` need to be `comb`

Description

By construction `tokens_a_withdrawn` is smaller than `token_pool_a`
(equal to `token_pool_a*lqt_burned/lqt_total` and we check that `lqt_burned < lqt_total`).
While it is a good practice to check posible negative value when casting from `int` to `nat`, when we already have that guarantee, you can simply replace `is_nat` by `abs`, which will reduce the gas cost.

Q8. Miscelanous readability improvement

Category	Impact	Location	Status
Suggestion	Readability	DEX	Acknowlegde

Description

The errors are all prefixed with `error_`. This is a good design pattern and it can be "automated" by taking adventage of module

```
1    module Error = struct
2        ...
3    end
```