LIGO

WUλFMAN CORPORATION

Audit LP-migration

March 2022

Tezos

Contents

Disclaimer

This report does not provide any warranty or guarantee regarding the absolute bug-free nature 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 Ekino

The object of the audit is the analysis of the LP-migrationin 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	LP-migration
Publisher	Ekino
Platform	Tezos
Language	Smartpy
Codebase	`https://github.com/Smartlinkhub/LP-Migration`
Original commit	fea7e52071f09fe97c910e1401fcc5e0463e9341
Contract address	
Contract URL	

Audit summary

Auditor	Wulfman Corporation
Delivery date	March 2022
Scope	Smartpy contracts
Methodology	Manual review
Tezos version	Hangzhou
Tezos client version	

Vulnerability summary

Total issues	5
Critical	0
Major	3
Medium	0
Minor	1
Informational	1

Code Quality summary

Total improvements	0
Maintenance	0
Scalability	0
Readability	0
Origination cost	0
Gas cost	0

Vulnerability

Contents

General Remark

The contract being use as a trusted 3^{rd} party to move asset from a first DEX (Quipuswap) to a new one (Vortex), its inner behavior remains quite straightforward while a specific care needs to be put in the contract to contract interactions and the possible issues that result in calling a malicious contract. According to the information ported to our attention and our understanding of `migration_quipuswap_vortex_fa2.py` and `migration_quipuswap_vortex_fa12.py`, the re-entrance issues related to these contracts are correctly handled, as long as the user call the contract through the Smartlink UI. (Except for the potential Token loss issue mentioned in section V2. when use with a malicious *Token* contract. We can argue that such a token will have no economic value. We let the reader decide if the loss of such token is an issue)

In case the user doesn't use the SmartLink UI (for instance, an attacker sending a malicious lookalike UI or doing a man in the middle attack, which will then inject wrong values for both DEX), some attacks become possible due to the different point indicated in the document. As we cannot guarantee how the smart-contract will be used, the severity level doesn't take into account that the UI is trustworthy. The proposed solutions should fix the vulnerabilities even in the case of malicious UI.

Notice that these re-entrance problems is link to the necessity of communicating with a call-back mechanism, which interrupt the normal flow of the algorithm and split it in several callable entry-point, opening a surface of attack. Interacting with contract that use the view mechanism instead (which should be the norm in the future) will avoid this issues

V1. Invalid re-entrance test for migrate entry-point

Category	Severity	Location	Status
Potential vulnerabilities	Major	migration_quipuswap_vortex_fa2.py	

//

Description

Migrate should be called only at the beginning of the call sequence. To do so, it checks that the migration data are `None`, or if they are not none. That the current source is different from the one stored in the storage.

For this last case to happen, you would need a first call to migrate, that triggers a chain reaction which will call the `balance_of` entrypoint of the *Token* contract. Then the contract will not use the callback to the *migration* contract, leaving it in an unstable state

An adversary could craft a token contract that will do so, and then send a transaction from another source account to reenter the contract. Thus, the guard of this entry-point does not prevent re-entrance. It only prevent that the reentrancy occurs in a single external transaction instead of several.

Solution

In order to prevent any re-entrance, the entry-point should fail when migration data are not `None` (or when state is `NO_MIGRATION`). However, this will allow to *DDoS* the contract with a similar attack. And this is probably the reason why the contract is currently not doing so.

To prevent this, this contract need a mechanism to clean the contract and put it back a safe state. This mechanism can be a separate entry-point. By doing so, an attacker can still regularly send transactions that will block the contract after a reset. Considering that it should not cause a security issue that anyone can reset the contract (instead of just an admin), the reset mechanism can be at the beginning of the `migrate` entry-point which stop the possibility of *DDoS*.

V2. Token loss

Category	Severity	Location	Status
Potential vulnerabilities	Major	migration_quipuswap_vortex_fa2.py	

Description

If the migration operation were to be incomplete (due to a call to an adversarial contract in an internal transaction similarly to the previous point V1.), but after the call to the transfer entry-point of the contract *Token*. Then the entry-point will have concluded, leading to tokens being move from a user account to the *Migration contract* account. Those token are now belonging to the *migration contract* which is not able to use them.

Solution

Similar to the previous section. The smart-contract should provide an entry-point to "fix" the contract in case if it ends up in an incoherent state. This entry-point will look into the value of `storage.stage` to detect which step have been executed before and needs to be reverted

V3. Stealing tokens

Category	Severity	Location	Status
Potential vulnerabilities	Major	migration_quipuswap_vortex_fa2.py	

Description

Following on the previous ideas. And on the fact that contract address are not hard-coded but given at each call to migration. An adversary could trick user to first do a migration with the right *Initial_Dex* contract and any *Token* contract but a malicious *Vortex* contract. This will have the effect to load the user address on the migration contract account. Then a call by the exploiter to the contract with the right *Vortex* contract will have the effect to move the token from the *migration contract* account to the exploiter account.

N.B. : This should not be an issue as long as the user go through the design IU and don't make the transaction itself

Solution

Instead of making both DEX address as parameter, hard-code them in the code or initialize them at origination. This is in most case a good practice as it reduce the surface of attack for the contract.

V4. Default Ownership to source may be an issue

Category	Severity	Location	Status
Potential bug	Minor	migration_quipuswap_vortex_fa2.py	

Description

The contract considers that the owner of the token to transfer is the source of the transaction. Which means the migration can only be done for user account and not for implicit accounts. It could be an issue if implicit accounts are registered on the DEX.

Solution

Allowing the owner to be any account given as a parameter seem to create other security issue but since we need to approve the migration contract before, it should be alright. The other solution would be to take the sender, but then the contract linked to the implicit account need to have an entry-point for the migration, which makes no sense.

In this end this may be a necessary limitation.

V5. Documentation of call sequence is ambiguous

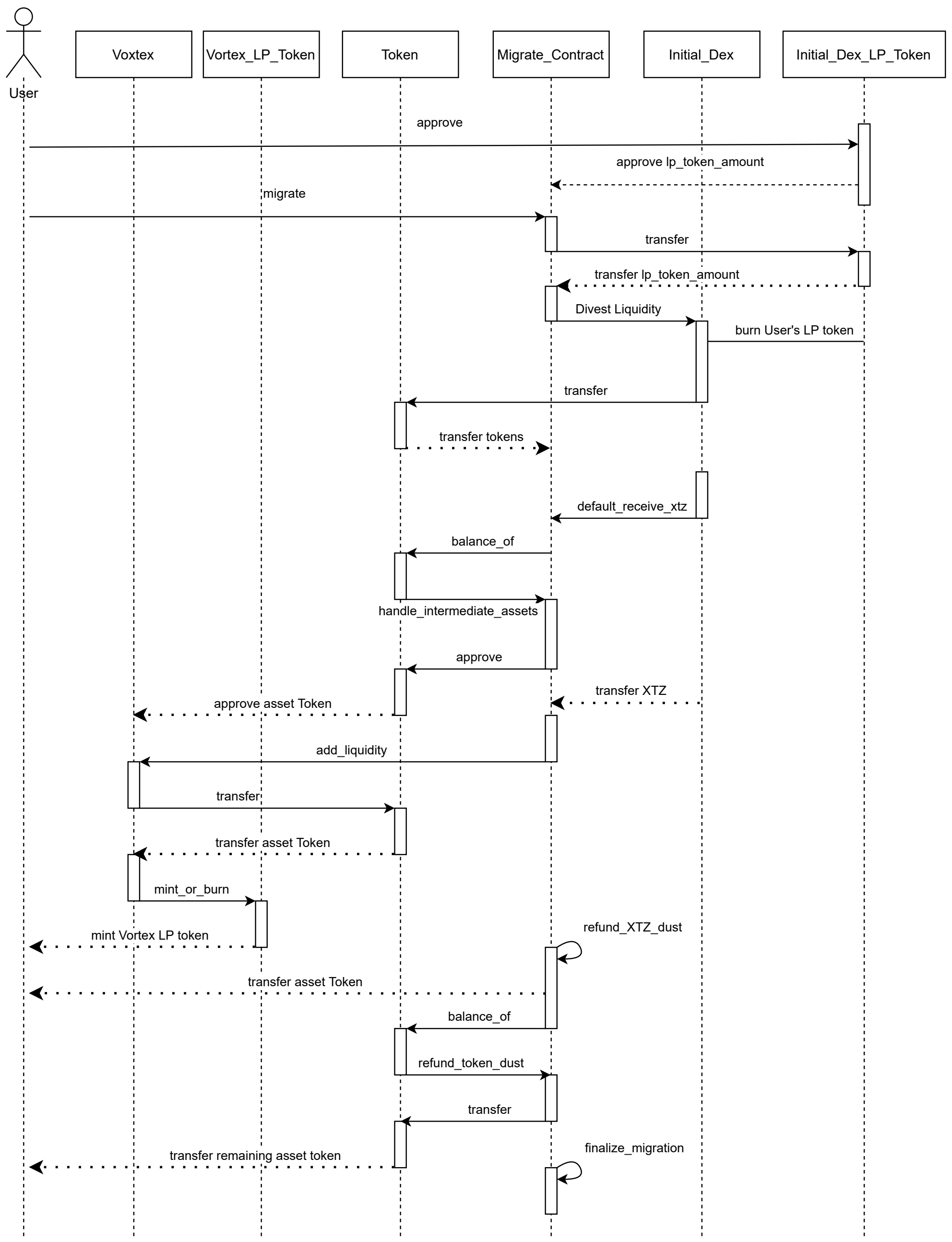
Category	Severity	Location	Status
Documentation	Informational	migration_quipuswap_vortex_fa2.py	

Description

The provided sequence diagram was appreciated. However, it is odd looking that the transfer operation doesn't start right after the transaction, as if contract execution were asynchronous, like for web servers. After further looking, it seems that the error was that the arrows represent the creation of the corresponding operations in the calling contract and not the call of this internal operation by the protocol, which is a bit odd.

Solution

I remade the diagram making the operation start right after the call. It is still not realistic as on my diagram there are additional processing after the call when normally all *internal operations* are emitted at the same time. I decided to add it for illustration and possible discussion on what would be the correct way to present the sequence diagram for Tezos smart-contract


User
Voxtex
Vortex_LP_Token
Token
Migrate_Contract
Initial_Dex
Initial_Dex_LP_Token
approve
approve lp_token_amount
migrate
transfer
transfer lp_token_amount
Divest Liquidity
burn User's LP token
transfer
transfer tokens
default_receive_xtz
balance_of
handle_intermediate_assets
approve
transfer XTZ
approve asset Token
add_liquidity
transfer
transfer asset Token
mint_or_burn
mint Vortex LP token
refund_XTZ_dust
transfer asset Token
balance_of
refund_token_dust
transfer
finalize_migration
transfer remaining asset token