Ubinetic's Youves engine

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 12.08.2022

Table of contents

Version / Date	Description
1.0 / 12.08.2022	Final version

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of Ubinetic's Youves engine.

Inference AG performed the security assessment based on the agreed scope, following our approach and activities as outlined in the "Project overview" chapter between 12th July 2022 and 12th August 2022. Feedback from Ubinetic was received and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our security assessment revealed several security issues. Additionally, different observations were also made which, if resolved with appropriate actions, may improve the quality of Ubinetic's Youves engine.

This report only shows remaining open or partly resolved issues and observations.

Overview on issues and observations

Details for each reported issue or observation can be obtained from the "Security issues" and "Observations" sections.

Row header	Severity / Status
Issues	
S-UYE-001: stakes registered in set data type	low / open
S-UYE-002: fixed stakes updates missing	low / open
S-UYE-003: Manipulation of interest rate using flash loans	medium / open
Observations	
O-UYE-001: Testing	- / open
O-UYE-002: FA2 non-conformance	- / open
O-UYE-003: Lack of documentation	- / open

O-UYE-004: AdministrableMixin	- / open
O-UYE-005: SingleAdministrableMixin	- / open
O-UYE-006: FA2 non-conformance in unified staking pool	- / open

Project overview

Scope

The scope of the security assessment was the following set of smart contracts (alphabetically ordered):

- Exchange oracle
- Governance token
- Interest rate updater (exponential)
- Interest rate updater (linear)
- Liquidity pool oracle
- Long staking pool
- Options listing
- Plenty oracle
- QuipuSwap oracle
- QuipuSwap token-to-token oracle
- Savings pool
- Stake manager
- Synthetic token
- Tez tracker engine
- Token options listing
- Token tracker engine
- Unified staking pool
- Vault
- Vester

Please see section scope details in the appendix for a more precise scope description.

All files in scope were made available via the source code repo https://gitlab.papers.tech/papers/ubinetic/sap-smart-contract and our initial security assessment considered commit “a705acd045b9cbed5e61cc8da0010326cf05eb1f”.

The following files from the source code repo were also part of the assessment:

- Testing files under folder /tests/tracker/ and /test/oracle and its included files in order to perform defined tests for the above mentioned in-scope smart contracts.

Additionally, we also had access to the following documentation in order to build up our understanding of the oracle protocol: https://docs.youves.com/

Our reassessment considered commit: a2f7b4771cbd77e39cf629af50402f5ed35bc660.

Scope limitations

Our security assessment is based on the following key assumptions and scope limitations:

- Any potential adversarial activities conducted by the administrator of the contract or operational errors by administrators were out of scope.
- Smart contracts for tokens not in scope were considered trusted.
- The economic model, including the appropriate settings for incentives, thresholds, factors, etc. was out of scope. Also out of scope was assessing the impact of changing incentives, thresholds, factors, etc. when the protocol is live by using an admin function.

Methodology

Inference's methodology for smart contract security assessments on Tezos is a combination of a source code review of the smart contract source code in the high-level language (e.g. Ligo or SmartPy), and the resulting compiled code in Michelson. This approach provides additional assurance that the compiler producing the Michelson code is correct, and does not introduce any security issues. Furthermore, this approach fosters a better understanding for smart contract users of what has been reviewed and what has been effectively deployed on-chain.

In order to ensure a high quality in our security assessments, Inference is using subject matter experts having a high adversarial scenario mindset to spot potential issues in smart contracts under review. Additionally, we apply checklists derived from good practices and commonly known issues based on the Tezos smart contract assessment checklist to document our work and to ensure good issue coverage.

Furthermore, Inference maintains regular communications with the smart contract development team to ensure a correct understanding of the smart contract solution and environment, but also to make teams aware of any observations as soon as possible.

Inference's internal quality assurance procedures ensure that results of security assessments are challenged for completeness and appropriateness by a second independent expert.

Objectives

The objectives are the identification of security issues with regards to the assessed smart contracts and their conceptual design and specification. The security assessment also focuses on adversarial scenarios on specific use cases which have been listed in appendix Adversarial scenarios. These were identified together with the Ubinetic developers and checked during our security assessment.

Activities

Our security assessment activities for the defined scope were:

- Source code review of smart contract code in SmartPy
- Source code review of the smart contracts in Michelson
- Review of testing code

We applied the checklist for smart contract security assessments on Tezos, version 1.1 obtained from https://github.com/InferenceAG/TezosSecurityAssessmentChecklist. We applied the following security checklist tables:

- System / Platform
- Storage
- Gas issues and efficiency
- Code issues
- Transactions
- Entrypoint
- On-chain views
- Admin / Operator functions
- Other topics & test cases

Our activities for the follow-up assessment were:

- Source code review of changes in the smart contract code in SmartPy
- Source code review of changes in the smart contracts in Michelson
- Reassessing security issues and observations from initial assessment in case they are claimed to be resolved

Security issues

S-UYE-001: stakes registered in set data type

When depositing tokens in the unified staking pools users can choose whether the deposited tokens are added to an existing “stake” or they can deposit them into a new “stake”.

Since the unified staking pool stores the user’s stake identifiers in a data structure of type “set”, this poses a risk that the loading of this set data structure fails due to gas exhaustion in cases where a user has created a huge amount of stakes.

Probability - Low. We rate this as low, since the creation of a huge amount of stakes is necessary.
Impact - Low. We rate this as low for the entire Youves engine protocol. However, the impact for the affected users may be large.
Severity - Low.

Recommendation:
We recommend analysing the situation and potentially considering elaborating a different storage architecture, which is not prone to gas exhaustion. At the very least, we recommend appropriately documenting this potential risk in the user-facing docs.

Comment from Ubinetic:
The frontend inhibits the creation of multiple stakes. CLI users should be redirected to this report.

S-UYE-002: fixed stakes updates missing

The concept of fixed stakes in the stake manager is used to assign a certain percentage of the total stake to specific addresses. Thus, whenever the total stake is changing, the fixed stakes have to change as well.

However, in the current implementation, the fixed stakes are not automatically adjusted when the total stake is changing.

Thus, depending on whether the total stake is decreasing or increasing, this is either a disadvantage or advantage for the user or the special addresses.

Probability - Low.
Impact - Low.
Severity - Low.

Additional note: After calling the admin restricted entrypoints “set_fixed_stakes” and “im” an update of the fixed stakes addresses should be done as well.

Recommendation:
We recommend analysing the situation and evaluating appropriate measures.

Comment from Ubinetic:
The fixed stakes are a list of addresses. In order to limit gas usage for all users, the update function has been outsourced to a specific entrypoint “update_fixed_stakes”, which is open for everyone.

In addition, a bot is regularly executing “update_fixed_stakes“.

S-UYE-003: Manipulation of interest rate using flash loans

Flash loans can potentially be used in order to push the interest rates to the defined possible maximum or minimum interest rates.

See also comments Interest rate updater (exponential & linear) in Adversarial scenarios.

Probability - Low.
Impact - Medium.
Severity - Medium.

Recommendation:
We recommend analysing the situation and evaluating appropriate measures.

Comment from Ubinetic:
This is monitored as a soft mitigation and the team is currently evaluating a solution with time-weighted price observations.

Observations

O-UYE-001: Testing

The smart contracts in scope have several defined test cases in SmartPy. However, for instance, we noted:

- No test cases were found which fully cover the AdministrableMixin and BaseFA2.
- No failing test cases checking:
  - whether non-admin can call stake_manger%import_stakes
  - whether non-admin can call stake_manger%set_stake_factor
  - whether non-admin can call stake_manger%set_fixed_stakes
- No test cases found for:
  - set_administrator, propose_administrator, and remove_administrator in unified_staking_pool
  - update_max_release_period, set_balance, balance_of, and update_trading_window in unified_staking_pool
  - Entrypoints in synth token smart contract
  - V3 tracker engines
- Missing test cases:
  - no checks found to verify whether someone other than internal can execute “internal_” entrypoints in savings_pool.
  - no checks found to verify whether someone other than “admin” (tracker_engine) can call set_delegate or withdraw in a vault contract.
  - failing test cases for withdraw and set_delegate entrypoints in vault smart contract

Please see also the Inference checklists provided about missing test cases for smart contracts and its entrypoints.
We also noted that for all of the contracts in scope there are no on-chain test cases available.

Recommendation:
We recommend defining and executing test cases in SmartPy appropriately covering all entrypoints. In addition to the testing in SmartPy, we also advise to define appropriate on-chain testing.

Comment from Ubinetic:
We are in the process with ECAD Labs to automate deployments and testing using Taqueria.

O-UYE-002: FA2 non-conformance

The unified staking pool smart contract is not compliant with the FA2 specification TZIP-012, since the contract does not have any token metadata.

Furthermore, the unified staking pool smart contract also does not provide contract-level metadata, which is recommended by the FA2 specification TZIP-012.

Recommendation:
We recommend adding token metadata.

Comment from Ubinetic:
It was never a requirement for the token to be FA2 compliant, but we took inspiration from the FA2 standard.

O-UYE-003: Lack of documentation

Technical documentation for the Youves engine makes it hard to easily get a basic understanding with the available documentation. For instance, we noted that the following documentation is lacking which does not indicate good-quality documentation:

- general overview of how the different smart contracts are connected and interact with each other,
- coding primitives defining how values have to be handled and exchanged between contracts, e.g. format of prices detailing how much precision they need to have,
- Notes of assumptions made.

For our security assessment of the Youves engine we were able to receive this information in various meetings with the Ubinetic developers. However, the lack of documentation poses a risk for the future of the Youves engine system, since knowledge may get lost or code is wrongly handled due to incorrect interpretations.

Recommendation:
We recommend improving the documentation.

Comment from Ubinetic:
Point taken. We are improving the documentation and it will be delivered to the 31st of August.

O-UYE-004: AdministrableMixin

The smart contracts governance_token, savings_pool, stake_manager, synth_token, and the tracker engines are using “AdministrableMixin”. The following observations with regards to the “AdministrableMixin” can be made:

1. No two-step procedure to add new admins,
2. All admins can be removed, which would make the contract no longer administrable.

Recommendation:
We recommend analysing the situation, ensuring with appropriate documentation and procedures that admins dealing with these contracts are not mistakenly adding non-working accounts or removing the last existing admin.

Comment from Ubinetic:
All contracts mentioned above are administered by a multisig contract and therefore the two step procedure is ensured by the multisig.

O-UYE-005: SingleAdministrableMixin

The SingleAdministrationMixin used by the long_staking_pool, unified_staking_pool, and exchange_oracle allows removing all admins, which would make the contract no longer administrable.

Recommendation:
We recommend analysing the situation, ensuring with appropriate documentation and procedures that admins dealing with these contracts are not mistakenly removing the last existing admin.

Comment from Ubinetic:
All contracts mentioned above are administered by a multisig contract and therefore the two step procedure is ensured by the multisig.

O-UYE-006: FA2 non-conformance in unified staking pool

The unified staking pool smart contract handles non-fungible tokens (NFT) and implements the FA2 standard TZIP-012. However, the unified staking pool does not incorporate token metadata information for these NFTs.

Furthermore, the unified staking pool contract does not follow the specification's recommendation regarding the storage structure.

Comment from Ubinetic:
It was never a requirement for the token to be FA2 compliant, but we took inspiration from the FA2 standard.

Disclaimer

Appendix

Scope details

- Exchange oracle
  - contracts/oracle/exchange_oracle.py and the included SmartPy files in order to compile the smart contract with the "ExchangeOracle" as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/oracle/all/ExchangeOracle/step_000_cont_0_contract.tz
- Governance token
  - contracts/tracker/governance_token.py and the included SmartPy files in order to compile the smart contract with the "GovernanceToken" as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/GovernanceTokeb/step_000_cont_0_contract.tz
- Interest rate updater (exponential)
  - contracts/tracker/interest_rate_updater_exponential.py and the included SmartPy files in order to compile the smart contract with the "InterestRateUpdaterExponential" as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/InterestRateUpdaterExponential/step_000_cont_0_contract.tz
- Interest rate updater (linear):
  - contracts/tracker/interest_rate_updater_linear.py and the included SmartPy files in order to compile the smart contract with the "InterestRateUpdaterLinear" as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/InterestRateUpdateLinear/step_000_cont_0_contract.tz
- Liquidity pool oracle:
  - contracts/oracle/liquidity_pool_oracle.py and the included SmartPy files in order to compile the smart contract with the "LPPriceOracle" and "FlippedLPPriceOracle" as a compilation target

- compiled contract in Michelson: __SNAPSHOTS__/compilation/oracle/all/LPPriceOracle/step_000_cont_0_contract.tz
- compiled contract in Michelson: __SNAPSHOTS__/compilation/oracle/all/FlippedLPPriceOracle/step_000_cont_0_contract.tz

- Long staking pool:
  - contracts/tracker/long_staking_pool.py and the included SmartPy files in order to compile the smart contract with the “FA1LongStakingPool” and “FA2LongStakingPool” as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/FA1LongStakingPool/step_000_cont_0_contract.tz
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/FA2LongStakingPool/step_000_cont_0_contract.tz
- Options listing:
  - contracts/tracker/options_listing.py and the included SmartPy files in order to compile the smart contract with the “OptionsListing” as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/OptionsListing/step_000_cont_0_contract.tz
- Plenty oracle:
  - contracts/oracle/plenty_oracle.py and the included SmartPy files in order to compile the smart contract with the “PlentyOracle” as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/oracle/all/PlentyOracle/step_000_cont_0_contract.tz
- QuipuSwap oracle:
  - contracts/oracle/quipuswap_oracle.py and the included SmartPy files in order to compile the smart contract with the “QuipuSwapOracle” as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/QuipuSwapOracle/step_000_cont_0_contract.tz

- QuipuSwap token-to-token oracle:
    - contracts/oracle/quipuswap_token_to_token_oracle.py and the included SmartPy files in order to compile the smart contract with the "QuipuSwapTokenToTokenOracle" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/QuipuSwapTokenToTokenOracle/step_000_cont_0_contract.tz
- Savings pool:
    - contracts/tracker/savings_pool.py and the included SmartPy files in order to compile the smart contract with the "SavingsPool" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/SavingsPool/step_000_cont_0_contract.tz
- Stake manager:
    - contracts/tracker/stake_manager.py and the included SmartPy files in order to compile the smart contract with the "StakeManager" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/StakeManager/step_000_cont_0_contract.tz
- Synthetic token:
    - contracts/utils/fa2.py and the included SmartPy files in order to compile the smart contract with the "AdministrableFA2" as a compilation target.
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/SyntheticAssetToken/step_000_cont_0_contract.tz
- Tez tracker engine
    - contracts/tracker/tez_collateral_tracker_engine.py and the included SmartPy files in order to compile the smart contract with the "TezCollateralTrackingEngine" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/TezCollateralTrackingEngine/step_000_cont_0_contract.tz
- Token options listing:
    - contracts/tracker/token_options_listing.py and the included SmartPy files in order to compile the smart contract with the "FA1OptionsListing", "FA1OptionsListingExtraPrecisionOracle", and "FA2OptionsListing" as a compilation target

    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/FA1OptionsListing/step_000_cont_0_contract.tz
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/FA2OptionsListing/step_000_cont_0_contract.tz
- Token tracker engine
    - contracts/tracker/token_collateral_tracker_engine.py and the included SmartPy files in order to compile the smart contract with the "FA2TrackerEngineV3" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/FA2TrackerEngineV3/step_000_cont_0_contract.tz
- Unified staking pool:
    - contracts/tracker/unified_staking_pool.py and the included SmartPy files in order to compile the smart contract with the "UnifiedStakingPool" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/UnifiedStakingPool/step_000_cont_0_contract.tz
- Vault:
    - contracts/tracker/vault.py and the included SmartPy files in order to compile the smart contract with the "Vault" as a compilation target
    - compiled contract in Michelson as it is compiled into the contract: __SNAPSHOTS__/compilation/tracker/all/TezCollateralTrackingEngine/step_000_cont_0_contract.tz
- Vester:
    - contracts/tracker/vester.py and the included SmartPy files in order to compile the smart contract with the "Vester" as a compilation target
    - compiled contract in Michelson: __SNAPSHOTS__/compilation/tracker/all/Vester/step_000_cont_0_contract.tz

Adversarial scenarios

The following adversarial scenarios have been identified together with the Ubinetic developers and checked during our security assessment.

Exchange oracle

Scenario	Impact rating & descriptive	Assessment result
Price from the configured price feeder is changing abruptly and no longer reflects the real actual price (price feeder failure or hack, flash loan manipulations, etc.)	High: Youves's unified staking pool receives wrong prices and will exchange unfavourably with respect to the unified staking pool users if the DEX used for exchange has a similar price which may be manipulated with a flash loan.	**Note** No controls are implemented in the exchange oracle smart contract. The price feeder used in the exchange oracle will be the “generic oracle v3* from Ubinetic. Please also see Inference’s security assessment report “Unbinetic’s oracle smart contracts”.

Governance token

Scenario	Impact rating & descriptive	Assessment result
Users can claim more governance token than their stake is.	Medium: Inappropriate distribution of assets.	**Ok** Nothing identified.

Interest rate updater (exponential & linear)

Scenario	Impact rating & descriptive	Assessment result
Manipulation of interest rate.	Medium: Wrong calculation of interest rate.	**Note** An interest rate recalculation can be triggered once every 24 hours (linear variant) or. 7 days (exponential variant) by calling the non-permissioned entrypoint “interest_rate_update”. With each recalculation the maximum change of the interest rate is limited by a “target_step”. Furthermore, the interest rate can be higher / lower than a defined maximum / minimum. However, users could manipulate via flash loans the DEX from where the observed prices are obtained and then call the “interest_rate_update”. This could be repeated every 24 hours (linear variant) or. 7 days (exponential variant) in order to push the interest rate into a certain direction (defined minimum or maximum).

Liquidity pool oracle

Scenario	Impact rating & descriptive	Assessment result
The calculated price is changing abruptly and no longer reflects the real actual price (flash loans manipulations, etc.)	Medium: Wrong calculation of interest rate.	**Note** The liquidity pool oracle has a protection to limit huge and abrupt changes in prices. See also adversarial scenario tests for Interest rate updater (exponential & linear)

Long staking pool

Scenario	Impact rating & descriptive	Assessment result
Preventing a reset of the “stake age” by adding in a first step a zero or small amount and in a second step a bigger amount allowing that the bigger amount can be withdrawn at any time, but getting all accumulated rewards.	Low: No direct financial impact to the protocol. However, expected overall incentives for the protocol are bypassed.	**Ok** Nothing identified.
Lowering the locking time period in order to bail out from staking, but getting all rewards (or more than only the time partially calculated reward)	Low: No direct financial impact to the protocol. However, expected overall incentives for the protocol are bypassed.	**Ok** Nothing identified.
Get a higher stake assigned than the amount deposited.	High: Loss of assets.	**Ok** Nothing identified.
Withdraw a higher amount than the equivalent for the assigned stake.	High: Loss of assets.	**Ok** Nothing identified.

Options listing & token option listing

Scenario	Impact rating & descriptive	Assessment result
An option can be executed before the defined deadline has expired.	Low: Option is executed with a vault selected by the option creator and not deliberately accepted by a user who wants to fulfil the option from their own will.	**Ok** Nothing identified.
The option creator is choosing a vault which is above the defined settlement ratio.	Low: Collateral ratio of the user where the option is executed increases. However, under certain market conditions, this may not be favourable for the specific user, since it may outweigh the received premium fee.	**Ok** Vaults above a defined settlement ratio can not be used to execute an option.
A higher than the defined premium fee can be obtained by a user	High: Unapproved transfer of funds.	**Ok** A higher fee is only possible in the case where the option creator is choosing a complete under-water account to execute the option. However, the option creator which has to pay the higher fee has to execute this option on his own.
Users can create/advertise options for other users	High: Unwanted selling assets with a premium fee.	**Ok** Nothing identified.
Options can be cancelled by other users	Low: No (direct) financial impact for a user, but the user may lose time.	**Ok** Nothing identified.
Assets in options can be stolen.	High: Loss of assets.	**Ok** Nothing identified.

Plenty oracle

Scenario	Impact rating & descriptive	Assessment result
The calculated price is changing abruptly and no longer reflects the real actual price (flash loans manipulations, etc.)	Medium: Wrong calculation of interest rate.	**Note** No protection in the oracle itself. However, the interest rate update contract has some protection in. See also adversarial scenario tests for Interest rate updater (exponential & linear)

QuipuSwap oracle

Scenario	Impact rating & descriptive	Assessment result
The calculated price is changing abruptly and no longer reflects the real actual price (flash loans manipulations, etc.)	Medium: Wrong calculation of interest rate.	**Note** No protection in the oracle itself. However, the interest rate update contract has some protection in. See also adversarial scenario tests for Interest rate updater (exponential & linear)

QuipuSwap token-to-token oracle

Scenario	Impact rating & descriptive	Assessment result
The calculated price is changing abruptly and no longer reflects the real actual price (flash loans manipulations, etc.)	Medium: Wrong calculation of interest rate.	**Note** No protection in the oracle itself. However, the interest rate update contract has some protection in. See also adversarial scenario tests for Interest rate updater (exponential & linear)

Savings pool

Scenario	Impact rating & descriptive	Assessment result
Users get are higher stake than they should actually get, when depositing synth tokens.	High: Loss of assets.	**Ok** Nothing identified.
Users can extract more synth tokens than the equivalent to their stake.	High: Loss of assets.	**Ok** Nothing identified.
Synth tokens in saving pool can be stolen.	High: Loss of assets.	**Ok** Nothing identified.

Stake manager

Scenario	Impact rating & descriptive	Assessment result
Influence the stake distribution to the advantage of a user and/or for the disadvantage of the protocol.	Medium: Inappropriate distribution of assets.	**Ok** Nothing identified allows to directly influence the distribution key. However, please take note of our reported security issues S-UYE-002: fixed stakes updates missing.

Synthetic token

Scenario	Impact rating & descriptive	Assessment result
FA2 smart contract with privileged mint / burn entrypoints for admins (tracker engines). See scenario tests for the tracker engines.		

Tez tracker engine / Token tracker engine

Scenario	Impact rating & descriptive	Assessment result
Users mint more than the equivalent covered with available collateral.	High: Stability of synthetic asset not ensured.	**Ok** Nothing identified.
Users can unwind their positions, but do not have to provide all required synth tokens.	High: Stability of synthetic asset not ensured.	**Ok** Nothing identified.
Tez locked/required as collateral can be extracted by the user.	High: Stability of synthetic asset not ensured.	**Ok** Nothing identified.
Users can prevent the liquidation of their under collateralized positions.	High: Stability of synthetic asset not ensured.	**Ok** Nothing identified.
Liquidation of sufficient collateralized positions.	Medium: Financial loss (fee) for the liquidated user.	**Ok** Nothing identified.
Obtaining a higher governance stake than the one correctly to be assigned.	Medium: Wrong distribution of governance tokens	**Ok** Nothing identified.

Unified staking pool

Scenario	Impact rating & descriptive	Assessment result
Preventing a reset of the "stake age" by adding in a first step a zero or small amount and in a second step a bigger amount allowing that the bigger amount can be withdrawn at any time, but getting all accumulated rewards.	Low: No direct financial impact to the protocol. However, expected overall incentives for the protocol are bypassed.	**Ok** Nothing identified.
Lowering the locking time period in order to bail out from staking, but getting all rewards (or more than only the time partially calculated reward)	Low: No direct financial impact to the protocol. However, expected overall incentives for the protocol are bypassed.	**Ok** Nothing identified.
Get a higher stake assigned than the amount deposited.	High: Loss of assets.	**Ok** Nothing identified.
Withdraw a higher amount than the equivalent for the assigned stake.	High: Loss of assets.	**Ok** Nothing identified.
Trick swap functionality to exchange tokens at an unfavourable price.	High: Loss of assets.	**Ok** Not possible due to the expected minimal token amount calculation obtained from a DEX-independent oracle smart contract. However, manipulation would be possible in case the market where the independent price oracle obtains the price is manipulated as well.

Vault

Scenario	Impact rating & descriptive	Assessment result
Entrypoints are restricted to the tez tracker engine. See scenario test for the tez tracker engine.		

Vester

Scenario	Impact rating & descriptive	Assessment result
Bypass the vester contract so that withdrawn synth tokens from the savings pool are instantly available to the user.	Low: No direct financial impact to the protocol. However, expected overall incentives for the protocol are bypassed.	**Ok** Nothing identified.
Unlocking of own vested tokens before the deadline has expired.		**Ok** Nothing identified.
Steal vested or already unlocked tokens from other users.	High: Loss of assets for users.	**Ok** Nothing identified.

Risk rating definition for smart contracts

Severities are quantified with two dimensions, roughly defined as follows, whereas the examples have to be regarded as indication only:

Probability of occurrence / materialisation of an issue
(bullets for a category are linked with each other with “and/or” condition.)

- Low:
  - A trusted / privileged role is required.
  - Contract may end up in the issue if other conditions, which are also unlikely to happen, are required.
- Medium:
  - A specific role or contract state is required to trigger the issue.
  - Contract may end up in the issue if another condition is fulfilled as well.
- High:
  - Anybody can trigger the issue.
  - Contract’s state will over the short or long term end up in the issue.

Impact:
(bullets for a category are linked with each other with “and/or” condition.)

- Low:
  - Non-compliance with TZIP standards
  - Unclear error messages
  - Confusing structures
- Medium:
  - A minor amount of assets can be withdrawn or destroyed.
- High:
  - Not inline with the specification
  - A non-minor amount of assets can be withdrawn or destroyed.
  - Entire or part of the contract becomes unusable.

Severity:

	Low impact	Medium impact	High impact
High probability	**High**	**Critical**	**Critical**
Medium probability	**Medium**	**High**	**Critical**
Low probability	**Low**	**Medium**	**High**

Glossary

Term	Description
Ligo	High level smart contract language. Website: https://ligolang.org/
Origination	Deployment of a smart contract
SmartPy	High level smart contract language. Website: https://smartpy.io/
TZIP	Tezos Improvement Proposal