Token Lock

Smart Contract Audit

coinspect

Token Lock

Smart Contract Audit

V22076

Prepared for Trustswap • March 2022

1. Executive Summary

In **March 2022, Trustswap** engaged Coinspect to perform a source code review of **Token Lock** smart contracts. The objective of the project was to evaluate the security of the smart contracts.

Coinspect's smart contract security experts reviewed the source code provided and developed proof-of-concept tests to confirm the validity of the issues identified. The high-risk vulnerabilities identified are due to the lack of input validation. Coinspect did not have access to documentation nor test suites during this engagement.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
3	2	2
Fixed	Fixed	Fixed
3	2	2

2. Assessment and Scope

The audit started on **March 7, 2022**, and was conducted on the https://github.com/v-Rizzy/token-lock-smart-contract-algorand repository. The last commit at the audit time is 5f41b5e2d1a92949467b5fd97e7d75ec10bc8043 dated November 2, 2021.

The sha256 hashes of the reviewed files are:

```
e60bad9b94ee5ada329671b13abd37bcfb7975b68bd1f496fd5e9f596b0b6109  escrow_account.py
6fee1713e8279f1fa95122a84058876e670e0948993c66ad6bbfd5d590811fbe  state_manager.py
```

The Token Lock smart contracts are used to lock tokens for a specific amount of time.

The system is composed of two TEAL programs written using pyTEAL:

1. An escrow logic signature used for holding the locked tokens.
2. A state manager contract account which keeps the state of the system.

On **July 1, 2022** Coinspect verified that the changes applied on commit **`6921a9b0bbaf9a0d264e4d5fb2dbb1522ada9c96`** of the **main** branch addressed the issues raised in the initial assessment.

3. Summary of Findings

Id	Title	Total Risk	Fixed
TKL-1	Attackers can steal escrow funds	High	✔
TKL-2	State application can be deleted	High	✔
TKL-3	Burn funds through excessive fees	High	✔
TKL-4	Rekeying or closing state contract	Medium	✔
TKL-5	Denial of service on transfer lock	Medium	✔
TKL-6	Possible key override	Low	✔
TKL-7	Dangerous key manipulation	Low	✔

4. Detailed Findings

TKL-1	Attackers can steal escrow funds	
Total Risk **High**	Impact High	Location `escrow_account.py`
Fixed ✔	Likelihood High	

Description

The `opt_in` function does not check neither for `RekeyTo` nor `AssetCloseTo`, allowing any algorand account to steal funds or take over the account.

Attackers can force the escrow account to close out with an arbitrary address in the `CloseRemainderTo` or `AssetCloseTo` field. This would cause the funds in the escrow account to be moved into the attacker selected address.

Similarly, adversaries can gain complete control of the `logicsig` (or its delegated account), abusing the lack of validation of the `RekeyTo` value.

```
And(
    Txn.type_enum() == TxnType.AssetTransfer,

    #Sender and receiver of asset transfer must be the same account
    Txn.sender() == Txn.asset_receiver(),
)
```

An attacker can exploit this issue by making an `AssetTransfer` transaction of any arbitrary `ASA`, sending `zero` tokens from the escrow to itself and passing an arbitrary `AssetCloseTo` or `RekeyTo` account.

Recommendation

Always check that `RekeyTo` and `AssetCloseTo` are zero or valid values.

Status

Escrow account was removed in favor of inner transactions.

TKL-2 State application can be deleted

Total Risk	Impact	Location
High	High	`state_manager.py`
Fixed ✓	Likelihood High	

Description

Attackers can delete the contract code. The state contract fails to check that it will not be deleted on completion in multiple paths. An example is the `on_update_lock_period`.

Any user can lock a minimum amount of tokens in the contract and delete it with a `onComplete.`DeleteApplication opcode.

Recommendation

Always check the `onComplete` property for all possible paths.

Status

Issue addressed by following recommendation.

TKL-3	Burn funds through excessive fees	
Total Risk High	Impact High	Location `escrow_account.py`
Fixed ✓	Likelihood High	

Description

The `Fee` property is never validated in the escrow account. Attackers can exploit this to burn all `ALGO` held by the contract.

Alternatively, it is possible for the escrow to time lock assets to self and withdraw them immediately abusing `on_deposit` and `on_withdraw`. Repeating these operations would burn all the `ALGO` available for fees.

Recommendation

Check that the fee value is bounded by a reasonable value and avoid time locking to self.

Status

Escrow account was removed in favor of inner transactions.

TKL-4	Rekeying or closing state contract	
Total Risk **Medium**	Impact High	Location `state_manager.py`
Fixed ✓	Likelihood Low	

Description

Users can be manipulated into signing transactions with the `RekeyTo` and `CloseRemainderTo` fields set and lose their account/ALGOs.

These transaction fields are not validated in multiple paths of the state manager:

- `on_update_lock_period`
- `on_transfer_lock_owner`
- `on_claim_lock_owner`
- `on_opt_in` and `on_closeout`

Recommendation

Make sure that these transaction fields are checked for all execution paths.

Status

Issue addressed by following recommendation.

TKL-5	Denial of service on transfer lock	
Total Risk Medium	Impact Medium	Location `state_manager.py`
Fixed ✓	Likelihood High	

Description

Attackers can stop future lock transfers. The transfer locks are saved on the global state, and given that it is limited, an attacker could fill it up to prevent other of the transfer lock functionality.

```
#Read global state to check for lock transfer in progress
def lock_transfer_status(account: Bytes):
      return App.globalGet(Concat(account, Gtxn[0].application_args[2]))
```

Users of the platform can fill the global state by successive calls to transfer `on_transfer_lock_owner` preventing any user to add values to the global state, in particular preventing any other `on_transfer_lock_owner` call.

Recommendation

Use local storage for `lock_transfers`.

Status

Issue addressed by following recommendation.

TKL-6	Possible key override	
Total Risk Low	Impact Low	Location `state_manager.py`
Fixed ✓	Likelihood Low	

Description

Users can lose funds because the `on_claim_lock_owner` function does not check if the key already exists. When a user calls `on_claim_lock_owner`, there is no check for the key to exist in the sender storage.

This can lead to loss of funds for the sender.

Recommendation

Verify that expected new keys do not exist before writing the values.

Status

Issue addressed by following recommendation.

TKL-7 Dangerous key manipulation

Total Risk	Impact	Location
Low	Low	`state_manager.py`
Fixed ✓	Likelihood Low	

Description

When the users make a deposit, a key is generated for that deposit given by the call arguments.

```
scratchvar_key.store(Concat(Itob(Gtxn[1].xfer_asset()), Gtxn[0].application_args[2],
            Itob(Gtxn[1].asset_amount()), Itob(Global.latest_timestamp()))),
```

This value is later written with the unlock timestamp:

```
write_user_local_state(
    scratchvar_key.load(),
    Btoi(Gtxn[0].application_args[1])
)
```

But nothing prevents scratchvar_key to hold a reserved value like "`TRUSTSWAP_ESCROW`" or "`COUNTER`".

Coinspect did not find any exploitation path for this issue, but key manipulation increases the attack surface of the contract.

Recommendation

Add a prefix for deposit keys.

Status

Issue addressed by following recommendation.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.