Secure Code Review

Findings and Recommendations Report Presented to:

AlgoRai

September 22, 2022
Version: 2.0

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

FOR PUBLIC RELEASE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

AlgoRai engaged Kudelski Security to perform a secure code assessment of smart contracts powering its ALGORAI FINANCE system.

The assessment was conducted remotely by the Kudelski Security Team.
Testing took place on August 5, 2022 - September 16, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered with the smart contracts.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Missing Authority Checks
- Lack of Input Validation
- Functional Bugs

Important note regarding all smart contracts and the way they are managed:

- The administrator requires a lot of trust given the amount of access they have, so any user is relying on appropriate security controls outside of the smart contracts.

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code was clean, concise, and commented throughout
- Tests were also provided as part of the project, which is convenient for better understanding the smart contracts and useful for elaborating scenarios and validating findings
- Finally, we had regular and very enriching technical exchanges on various topics.

Scope and Rules Of Engagement

Kudelski performed a Secure Code Review for AlgoRai. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied with the commit hashes in private repositories at:

- `https://gitlab.com/algo-foundry/algorai/vaults/-/commit/433312aacf11dfe126d104782271bd5821d43014`
    - Subfolder `contracts`
    - Written with Teal version 6
- `https://gitlab.com/algo-foundry/algorai/price-feeder/-/commit/bc8e1243251f6860bf0402b38f268a84d7114bc4`
    - Subfolder `deploy/src/contracts`
    - Written with PyTeal

A further round of review was performed by Kudelski Security, September 15-16, 2022, on remediations with the commit hashes at:

- `https://gitlab.com/algo-foundry/algorai/vaults/-/commit/b00f5032e7e4a7f3d4ffc8b0a4d27afdb8bf6d77`
- `https://gitlab.com/algo-foundry/algorai/price-feeder/-/commit/656f619abe4b366d02f93860cc200aa6f8623385`

In-Scope Contracts	
Vaults	Price-Feeder

`contracts/`
`├── blackList_whiteList.teal`
`├── clear.teal`
`├── deposit.teal`
`├── vault.teal`
`└── withdraw_round.teal`
	`deploy/src/contracts`
	`├── contracts.py`
	`├── medianizer_contract.py`
	`└── methods.py`

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we discovered 2 findings that had a high severity rating, as well as 6 of medium severity.

The following chart displays the findings by severity.

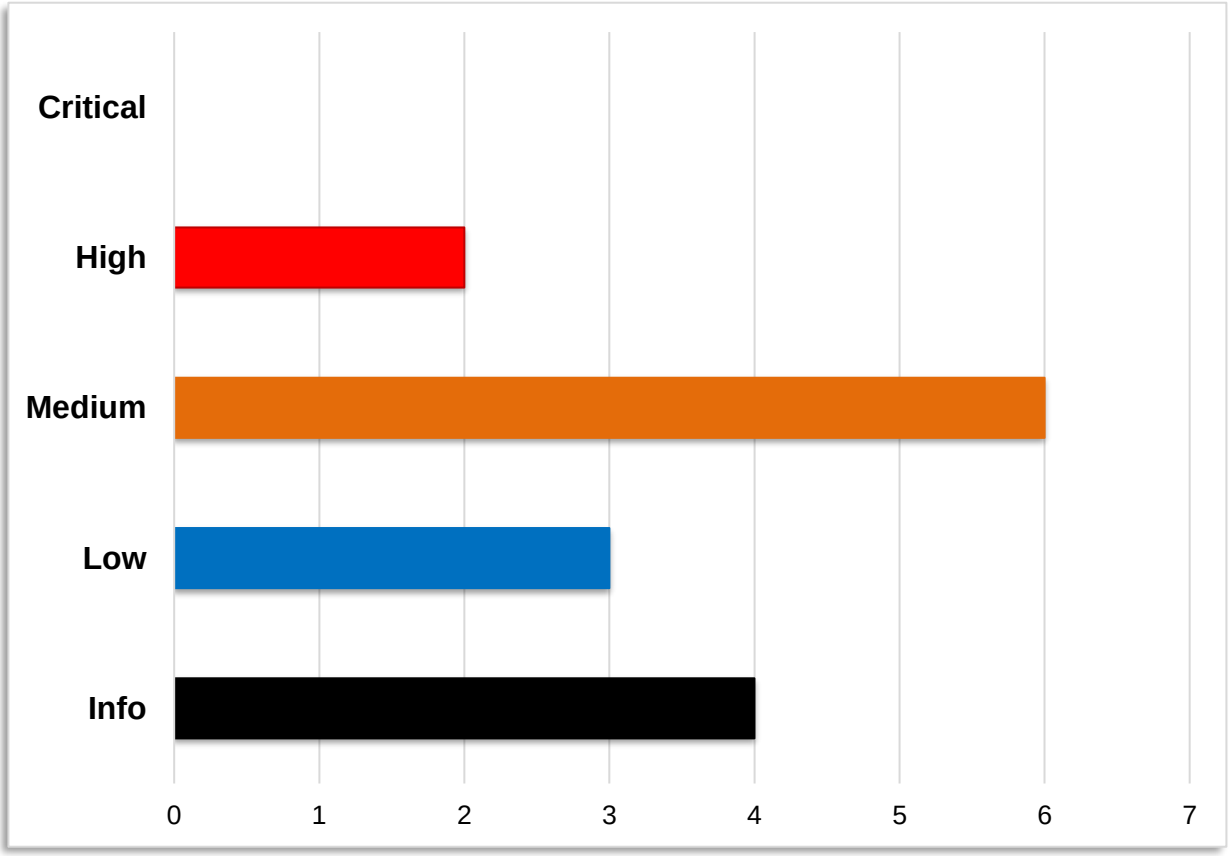


Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description	Status
KS-AR-01	**High**	Black/White list no update possible for addresses with index larger than 10	**Resolved**
KS-AR-02	**High**	Smart Contract life cycle not fully implemented	**Resolved**
KS-AR-03	**Medium**	Incorrect check for CurrentApplicationID	**Resolved**
KS-AR-04	**Medium**	Cancel Withdraw performs no checks on the other transaction in a group	**Resolved**
KS-AR-05	**Medium**	Lack of security checks on admin-called functions	**Resolved**
KS-AR-06	**Medium**	Missing configuration or parameter checks	**Resolved**
KS-AR-07	**Medium**	CloseOut accepting transactions while the user still has funds	**Resolved**
KS-AR-08	**Low**	Group transactions initiated by the admin do not check the Sender of the second transaction	**Resolved**
KS-AR-09	**Medium**	Performing mathematical operations with very low amounts	**Resolved**
KS-AR-10	**Low**	Black/White list no duplicate check	**Resolved**
KS-AR-11	**Low**	TWAP value error management	**Resolved**
KS-AR-12	**Informational**	Incorrect int to ASCII conversion in blackList_whiteList	**Resolved**
KS-AR-13	**Informational**	Mismatches between the documentation, comments, and the code	**Informational**
KS-AR-14	**Informational**	Code duplication	**Resolved**
KS-AR-15	**Informational**	Inconsistent naming conventions	**Informational**

Table 2: Findings Overview

KS-AR-01 – Black/White list no update possible for addresses with index larger than 10

Severity	HIGH
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	High	Easy

Description

The `blacklist_whitelist.teal` contract keeps track of users that can either

a) perform a deposit while the contract is paused (whitelist) or

b) are never allowed to perform a deposit (blacklist)

The smart contracts allow users to be added to these lists. The addresses stored in these contracts can be modified by an administrator with authorization to run these contracts, in case a user needs to be removed from one of these lists.

However, in the reviewed implementation of this smart contract, the update procedure was only possible for addresses stored from the index 1 to 9, while the contracts can store up to 62 such addresses. Therefore, the addresses from index 10 to 62 stored in these contracts, once created, can never be modified.

Impact

A user that has been added to blacklist cannot perform deposits. If their address in the blacklist has an index greater than 10 then it is impossible to remove them from the list and this user will never be able to perform a deposit in the AlgoRai vault. This represents a denial of service and could result in reputational damage to AlgoRai.

Evidence

```
181     load 0
182     int 49 // addr_1
183     >=
184     &&
185     load 0
186     int 57 // addr_9
187     <=
188     &&
189     bz fail
```

Valid range is only from 1 to 9

Affected Resource

- `vaults/contracts/blackList_whiteList.teal (Lines 181-189)`

Recommendation

Fix the code to allow addresses with an index between 10 and 62 to be updated.

Reference

N/A

KS-AR-02 – Smart Contract life cycle not fully implemented

Severity	HIGH
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Medium	Easy

Description

In the price-feeder contracts a "*tip"* feature was implemented to "*provide a reward to the reporter for reporting*". We observed that the nominal case was well implemented via the two calls/methods `tip` and `report`. But in case of closure for instance, we observed that nothing was implemented about the possible remaining *tip* amount still present in the smart contract.

Impact

Depending on values such as `CloseOut/CloseRemainderTo/RekeyTo`, the remaining tip amount may be lost.

Evidence

```
20      program = Cond(
21          [Txn.application_id() == Int(0), create()],
22          [Txn.on_completion() == OnComplete.NoOp, handle_method()],
23          [Txn.on_completion() == OnComplete.DeleteApplication, Return(is_governance)],
24          [Txn.on_completion() == OnComplete.UpdateApplication, Return(is_governance)],
25          # [Txn.on_completion() == OnComplete.CloseOut, close()]
26      )
```

CloseOut branch is commented out.

Note that we have a "tip" app call in the contract, but we won't be using any tip functionality right now.

Comment present in the code.

Affected Resource

- `price-feeder/deploy/src/contracts/contracts.py (Line 25)`
- `Tellor Reporter Golang App.pdf`, page 3

Recommendation

Either completely implement the functionality or remove the code from the application.

Reference

N/A

KS-AR-03 – Incorrect check for CurrentApplicationID

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	Medium	Easy

Description

To be consistent with Algorand best practices each function in the smart contracts needs to perform a series of security and correctness checks on the incoming transactions. One of these checks verifies that the transaction (if it is an application call) is indeed targeting the current smart contract: `ApplicationID == CurrentApplicationID`. In some parts of the code, this check was not performed. Instead, another equality check was performed, between `gtxn x ApplicationID` and `txn ApplicationID`.

The equality check mentioned above will always return true, as it compared an element to itself. In addition, the correct check was not performed.

Impact

Multiple contracts of the same type (vault, deposit…) might simultaneously be deployed at the same time on the Algorand net. Without an explicit check on the intended Application ID, there might be no way to distinguish between transactions meant for different contract instances. We will illustrate one possible consequence with an example. Let us assume there exists two different instances of a deposit contract, deposit A and deposit B. Deposit A is linked to a vault with a call strategy. Deposit B is almost identical to A, but linked to a vault with a put strategy. In the absence of this check, a user might send a transaction meant for deposit B, have it accidentally accepted by deposit A, and as a consequence lose his money in an unintended way.

Evidence

deposit.teal

```
479  pause:
480      // verify that the creator or a
481      callsub admin_verify
482
483      // Pause is one tnx
484      global GroupSize
485      int 1
486      ==
487      gtxn 0 ApplicationID
488      txn ApplicationID
489      ==
490      &&
491      bz fail
```

withdraw_round.teal

```
597      ==
598      &&
599      // this tnx is App call
600      txn TypeEnum
601      int appl
602      ==
603      &&
604      // calling this contract
605      txn ApplicationID
606      global CurrentApplicationID
607      ==
608      &&
609      // verify first app passed i
```

On the right, an example of a correct check performed in `withdraw_round.teal`. On the left, an example of an incorrect check performed in `deposit.teal`. In this case, there exists only one transaction in the group, therefore `gtxn 0` and `txn` will refer to the same transaction.

Affected Resource

The issue manifests itself at the following points in the smart contracts:

- `vaults/contracts/deposit.teal (lines 487-489)`
- `vaults/contracts/deposit.teal (lines 581-583)`
- `vaults/contracts/deposit.teal (lines 982-983)`
- `vaults/contracts/withdraw_round.teal (lines 310-312)`

We note that in the case of the vault contract, this is not an actual security issue. At the lines of code listed below, *both* checks are performed. Thus, this finding is not present. We enumerate the relevant lines here for completeness.

- `vaults/contracts/vault.teal (lines 354-359)`
- `vaults/contracts/vault.teal (lines 493-498)`

Recommendation

Where applicable in `deposit.teal` and `withdraw_round.teal`, replace the code with the correct check.

Reference

N/A

KS-AR-04 – Cancel Withdraw performs no checks on the other transaction in a group

Severity	Medium
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Medium	Easy

Description

This concerns the `Cancel Withdraw` procedure, which a user may call to cancel a withdrawal, thus moving their funds from the `Withdraw` contract back to `Vault` contract. When a user performs a `Cancel Withdraw`, they send a group of two transactions:

1. The first transaction is an application call to the `withdraw_round.teal` requesting to move the Algos/Assets back to the `vault.teal` contract
2. The second transaction is an application call to the deposit contract to reinstate the local state representing the share of the user.

Each contract then performs security checks on its transaction. However, there were no security cross-checks performed on all apart from the `deposit.teal` contract, where a check was performed to ensure the `Sender` is the same for both transactions.

Impact

The correctness of the state across all three contracts (`deposit.teal`, `vault.teal`, `withdraw_round.teal`) depends on whether both transactions get executed or both transactions fail. If only one transaction is accepted but not the other, this can create a state desynchronization.

For example, assume that a user sends a legitimate transaction meant for one of the contracts (e.g. `withdraw_round.teal`), then groups it with another transaction meant for some other smart contract (that approves it). As a possible consequence, the user might lose ownership over their share.

Evidence

See next page.

Affected Resource

- `vaults/contracts/deposit.teal (lines 1379-1434)`
- `vaults/contracts/withdraw_round.teal (Lines 573-652)`

Recommendation

One way of mitigating this would be to have each contract performs checks on the other transaction in the group, at a minimum:

- That the other transaction is an application call, and that the `ApplicationID` is set to the other contract. It may be the case that the deposit contract might need to check the global state of the vault contract, in order to determine the Address of the withdrawal contract.
- The correctness of `NumAppArgs` and `ApplArgs`, as required by the `handle_cancel_withdraw` from the other contract. This is to ensure that the transaction is validated by the correct sub-routine.
- As an additional security measure, that important fields such as `RekeyTo` and `CloseRemainderTo` are correctly set to the `ZeroAddress`.

Reference

N/A

Evidence

deposit.teal

```
1379  handle_cancel_withdraw:
1380      // tnx verifications
1381      // verify group size
1382      // must be two txns
1383      global GroupSize
1384      int 2
1385      ==
1386      // verify that this txn is second statef
1387      txn GroupIndex
1388      int 1
1389      ==
1390      &&
1391      // this tnx is App call
1392      txn TypeEnum
1393      int appl
1394      ==
1395      &&
1396      // calling this contract
1397      txn ApplicationID
1398      global CurrentApplicationID
1399      ==
1400      &&
1401      // ensure first txn sender is same as cur
1402      gtxn 0 Sender
1403      txn Sender
1404      ==
1405      &&
1406      bz fail
1407
1408      // Security checks
1409      // no rekeying txn
1410      gtxn 1 RekeyTo
1411      global ZeroAddress
1412      ==
1413      // no account close txn
1414      gtxn 1 CloseRemainderTo
1415      global ZeroAddress
1416      ==
1417      &&
1418      bz fail
1419
1420      // read deposit amount to update user's l
1421      // first transaction's 1st slot of scratc
```

withdraw_round.teal

```
588      // 1. tnx verifications
589      // verify group size
590      // must be two txns
591      global GroupSize
592      int 2
593      ==
594      // verify that this txn is first stateful
595      txn GroupIndex
596      int 0
597      ==
598      &&
599      // this tnx is App call
600      txn TypeEnum
601      int appl
602      ==
603      &&
604      // calling this contract
605      txn ApplicationID
606      global CurrentApplicationID
607      ==
608      &&
609      // verify first app passed in app array
610      byte "vid"
611      app_global_get
612      txn Applications 1
613      ==
614      &&
615      // verify fee amount
616      // this call triggers two inner txns (a
617      txn Fee
618      int 3000
619      >=
620      &&
621      bz fail
622
623      // Security checks
624      // no rekeying txn
625      gtxn 0 RekeyTo
626      global ZeroAddress
627      ==
628      // no account close txn
629      gtxn 0 CloseRemainderTo
630      global ZeroAddress
```

Side-by-side comparison of the security checks in handle_cancel_withdraw, both in deposit.teal and withdraw_round.teal. The deposit.teal contract perform no checks on the transaction at index 0 (beyond the Sender), while the withdraw_round.teal contract performs no checks on the transaction at index 1.

To better illustrate the possible impact of this finding, we have implemented a very simple use-case scenario. We show that, at a minimum, the lack of checks could lead to the user losing ownership over their share of the vault.

To illustrate our scenario, we have considered two users participating in AlgoRai. We have performed the following steps.

1. We have created two users: ACCOUNT1 and ACCOUNT2.
2. Before the start of the first round, both users perform a deposit of 1,000,000 MicroAlgos
3. The admin starts a vault round with spot price = 40000 and strike price = 50000
4. ACCOUNT1 requests to withdraw their deposit.
5. ACCOUNT1 cancels withdraw but we replaced the second transaction in the group with an app call to a contract which auto-accepts all transactions.

We will present the detailed states after performing steps 3, 4 and 5. The states after steps 1 and 2 are correct and not noteworthy, therefore we have omitted them.

After step 3, the deposit present in the deposit contract has been moved to the vault contract. The shares of both users are still stored into local state of the deposit contract for ACCOUNT1. The withdraw contract for this round has been created. In the following screenshots we illustrate the global state and the local state in the relevant contracts.

```
Vault
Debugger listening on ws://127.0.0.1:55342/0232a295-567d-4eee-be75-b04405169ea7
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
AppID:  9
GlobalState:  [
  osa: 0,     spp: 40000,
  ad: 0,      dc: 0,
  liv: 0,     st: 1,
  strb: 70,   did: 8,
  stp: 50000, rd: 1,
  wid: 20,    vtv: 2000000,
  tra: 0,     vtd: 2000000
]
Waiting for the debugger to disconnect...
```

Global state of the vault after step 3. All variables of the global state are correct; the vault contains the deposits of the two users, and vtv = 1000000+ 1000000. Note vtv = Vault Total Value and represents the total value of the funds within the vault contract

```
Withdraw
Debugger listening on ws://127.0.0.1:55348/eb86924e-6479-4c35-94f8-9210af75450a
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
AppID:  20
GlobalState:  [ dc: 0, vid: 9, vrd: 1, wst: 0 ]
Waiting for the debugger to disconnect...
```

Global state of the withdraw contract after step 3. All variables of the global state are correct.

```
root@29dabc08d357:~/testnetwork/Node# goal app read --app-id $DEPOSIT_ID --guess-format --local --from $ACCOUNT1
{
  "d": {
    "tt": 2,
    "ui": 1000000
  },
  "dt": {
    "tt": 2,
    "ui": 1000000
  },
  "vr": {
    "tt": 2,
    "ui": 1
  }
```

Local state of the deposit contract for user ACCOUNT1 after step 3.
All variables of the local state are correct:
the user ACCOUNT 1 has a corresponding deposit share of 1000000.

```
root@29dabc08d357:~/testnetwork/Node# goal app read --app-id $WITHDRAW_ID --guess-format --local --from $ACCOUNT1
{}root@29dabc08d357:~/testnetwork/Node#
```

Local state of the withdraw contract after step 3.
As nothing has been initialized yet, this is the expected output at this step (an empty state).

We now perform step 4: ACCOUNT1 requests to withdraw their deposit. This is done using the following JavaScript command:

```
src/js/WithdrawRequest.js <VAULT_ID> <USER_MNEMONIC>.
```

This command will then move the money belonging to the user ACCOUNT1 to the withdraw contract. This means that the global state variable will be updated with the share of ACCOUNT1 and the local variable will also be updated with their share. In the following screenshots we illustrate the global state and the local state in the relevant contracts in this point in time.

```
Vault
Debugger listening on ws://127.0.0.1:57812/6c1b1e30-2a63-443d-9abe-f150713db8a1
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
AppID:  9
GlobalState:  [
  st: 1,        ad: 0,
  osa: 0,       spp: 40000,
  dc: 0,        rd: 1,
  did: 8,       stp: 50000,
  liv: 0,       vtd: 2000000,
  tra: 0,       vwd: 1000000,
  vtv: 2000000, strb: 70,
  wid: 20
]
Waiting for the debugger to disconnect...
```

Global state of the vault after step 4. All variables of the global state are correct.
The vault contains a vtd = 2000000, out of which 1000000 has been withdrawn.

```
Withdraw
Debugger listening on ws://127.0.0.1:57818/246d6695-89d4-4700-bec4-22572d707ccb
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
AppID:  20
GlobalState:  [ vrd: 1, dc: 0, vid: 9, wra: 1000000, ws: 1000000, wst: 0 ]
Waiting for the debugger to disconnect...
```

Global state of the withdraw contract after step 4. All variables of the global state are correct, and the withdraw contract stores the 1000000 that has just been withdrawn.

```
}root@29dabc08d357:~/testnetwork/Node# goal app read --app-id $DEPOSIT_ID --guess-format --local --from $ACCOUNT1
{
  "d": {
    "tt": 2
  },
  "dt": {
    "tt": 2
  },
  "vr": {
    "tt": 2
  }
```

Local state of the deposit contract after step 4. All variables of the local state are correct. They are set to 0 as ACCOUNT1 has no share in the vault after requesting the withdraw.

```
}root@29dabc08d357:~/testnetwork/Node# goal app read --app-id $WITHDRAW_ID --guess-format --local --from $ACCOUNT1
{
  "da": {
    "tt": 2,
    "ui": 1000000
  },
  "wa": {
    "tt": 2,
    "ui": 1000000
  }
```

Local state of the withdraw contract after step 4. All variables of the local state are correct. The local state presents the amount of 1000000 microAlgos that ACCOUNT1 withdrew from the vault.

We can now perform step 5: ACCOUNT1 cancels their withdraw. The procedure requires a group of two transactions. Normally, the first one is an app call meant for the withdraw contract and second one is a transaction sent to the deposit contract in order to reinstate the local state of ACCOUNT 1. However, we have replaced the second transaction of the group of transaction with an app call to a dummy contract which outputs always one (and thus accepts all transactions). Then we performed the following executions:

- We generated a correct app call to WITHDRAW_ID to move the microAlgos of ACCOUNT1 back the vault
- We generated an app call to a dummy contract TRUE_ID that accept every transaction

The exact commands are presented in the Figure below:

```
goal app call --fee 3000 --app-id ${WITHDRAW_ID} --app-arg "str:cancel_withdraw" --from ${ACCOUNT1} --foreign-app ${VAULT_ID}
--app-account ${VAULT_ESCROW} --foreign-asset $DC  --out=unsgined_txn1.tx

goal app call --app-id ${TRUE_ID} --app-arg "str:add-addr" --app-arg "addr:"${ACCOUNT2} --from ${ACCOUNT1} --out=unsgined_txn1.tx

cat unsgined_txn1.tx unsgined_txn2.tx > combinedtransactions.tx
goal clerk group -i combinedtransactions.tx -o groupedtransactions.tx
goal clerk split -i groupedtransactions.tx -o split.tx

goal clerk sign -i split-0.tx -o signout-0.tx
goal clerk sign -i split-1.tx -o signout-1.tx
cat signout-0.tx signout-1.tx > signout.tx

goal clerk rawsend -f signout.tx
```

Attack command lines. The commands presented here were executed and accepted by the network.

After this step, the global and local states of the impacted contracts are as follows:

```
Vault
Debugger listening on ws://127.0.0.1:56880/51d91930-2371-456d-ae70-5e75ba05ac32
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
AppID:  9
GlobalState:  [
  liv: 0,      did: 8,
  ad: 0,       osa: 0,
  dc: 0,       rd: 1,
  vtd: 2000000, st: 1,
  stp: 50000,  tra: 0,
  spp: 40000,  vtv: 2000000,
  strb: 70,    vwd: 0,
  wid: 20
]
Waiting for the debugger to disconnect...
```

Global state of the vault after step 5. All variables of the global state are correct. Notice that the `vwd`, which represents the amount withdrawn from total deposit for the current round, is now equal to 0, which is consistent with a cancelled withdraw.

```
Withdraw
Debugger listening on ws://127.0.0.1:56886/736cd145-a4fd-4770-b978-ca900ac43668
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
AppID:  20
GlobalState:  [ dc: 0, vid: 9, wra: 0, ws: 0, vrd: 1, wst: 0 ]
Waiting for the debugger to disconnect...
```

Global state of the withdraw contract after step 5. All variables of the global state are correct. No assets are currently stored in this contract.

```
root@29dabc08d357:~/testnetwork/Node# goal app read --app-id $DEPOSIT_ID --guess-format --local --from $ACCOUNT1
{
  "d": {
    "tt": 2
  },
  "dt": {
    "tt": 2
  },
  "vr": {
    "tt": 2
  }
```

Local state of the deposit contract after step 5 for the user ACCOUNT1. Notice that now the values are incorrect. The local variables d and dt should be 100000 microAlgos instead of empty.This is because that following the Cancel Withdraw, the share of the user is now back in the vault. At this point, the user ACCOUNT1 has effectively lost their share and has no means of retrieving it from the smart contracts.

```
}root@29dabc08d357:~/testnetwork/Node# goal app read --app-id $WITHDRAW_ID --guess-format --local --from $ACCOUNT1
{
  "da": {
    "tt": 2
  },
  "wa": {
    "tt": 2
  }
```

Local state of the withdraw contract after step 5 for the user ACCOUNT1. The Algos belonging to ACCOUNT1 are empty as well. This means that ACCOUNT1 has no Algo in the withdraw contract. This is correct. Unfortunately for the user, it also means they have no mean of retrieving their assets, as the withdraw has taken effect and their share is now back in the vault.

After performing these five steps, ACCOUNT1's local states in the deposit and withdraw contracts are set to 0, which result in the loss of ACCOUNT1's microAlgos. Indeed, ACCOUNT1 cannot request withdraw anymore as according to local state of the deposit contract ACCOUNT1 does not have any active deposit in the vault. Additionally, ACCOUNT1 cannot withdraw their microAlgos from the withdraw contract because their microAlgos were suppressed from the withdraw contract as proven by the global state of the withdraw contract.

KS-AR-05 – Lack of security checks on admin-called functions

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Easy

Description

Whenever a function is called by an admin, there were a few security checks that were not being performed on certain transaction fields such that `TypeEnum`, `RekeyTo`, `CloseRemainderTo`, `CloseAssetTo`. This can potentially become an issue in the case of loss or compromise of admin credentials, or in case of an accidental erroneous command being sent to a deployed contract.

Impact

As the application is centralized by design, the admin has full control over critical parts of the system. We acknowledge that this is the intended architecture. In this situation it is important to limit, to the fullest extent possible, the impact of the admin credentials being compromised. For instance, in the absence of security checks such as `RekeyTo` and `CloseRemainderTo`, the party in possession of the admin credentials could potentially redirect the contents of the smart contract to an account of their choice or perform other high-impact actions.

Evidence

See next page

Affected Resource

- `vaults/contracts/vault.teal`
- `vaults/contracts/deposit.teal`

Recommendation

Unless this is an intended functionality, any admin-only functions should also have security checks such as `RekeyCloseTo`, `CloseRemainderTo`, `CloseAssetTo`. If this functionality should be part of the business logic, the access to it should be well-controlled and not enabled by default in most, if not all admin-called functions. For example, perhaps make it so that only the `CreatorAddress` can perform these operations, and that they are not available to any other administrator. This is only one possible remediation, and it is important that appropriate security controls are implemented outside the code regarding the management of the administration role.

Reference

N/A

Evidence

```
pause:
    // verify that the creator or admin is making the call
    callsub admin_verify

    // Pause is one tnx
    global GroupSize
    int 1
    ==
    gtxn 0 ApplicationID
    txn ApplicationID
    ==
    &&
    bz fail

    // pause value
    txna ApplicationArgs 1
    btoi
    store 0

    // verify pause value, should 0 or 1
    load 0
    int 0
    ==
    load 0
    int 1
    ==
    ||
    bz fail

    // get current state of pause.
    // if state is same as requested change, then nothing to change
    byte "dp"
    app_global_get
    load 0
    !=
    bz fail

    // save new value
    byte "dp"
    load 0
    app_global_put

    b finish
```

Example of an admin-called function without any security checks like TypeEnum, RekeyTo, CloseRemainderTo or CloseAssetTo.

KS-AR-06 – Missing configuration or parameter checks

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Easy

Description

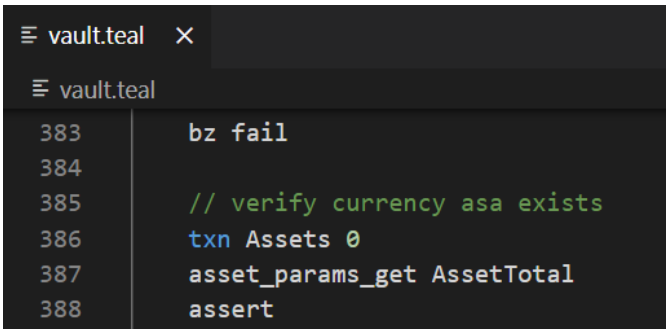
In the AlgoRai system the administrator capabilities are significant. As such whenever a function is called by an *admin*, it is important that certain checks are in place to ensure correct functionality. However, there were several checks that are not being performed such as:

- Order-of-steps checks: The functions in a smart contract should check, for example, that the contract has been initialized before performing any other operation.
- Parameter correctness checks: We noticed that the validity and correctness of the inputs given by the *admin* were not verified. The most critical ones we found were:
  - At the start of each round, the *admin* provides the strike and spot prices. Both prices must not be equal. This check between spot and strike prices was not done.
  - When executing the settlement pending procedure, the *admin* provides the settlement price which needs to be greater than 0. This check was not performed.

Impact

To a degree, the state of the system depends on it being correctly deployed and configured by the admin. While the likelihood is low, the risk of an incorrect configuration can lead to the system ending in an unstable state. For example, if a vault round is started with the spot price equal to the strike price, as described above, the vault round will not end (there is no code to capture this case). In this case all participants would lose control over their funds.

Evidence



Example of a check that is performed in vault.teal and withdraw_round.teal, but not in deposit.teal.

Affected Resource

- `vaults/contracts/vault.teal`
- `vaults/contracts/deposit.teal`
- `vaults/contracts/withdraw_round.teal`
- `vaults/contracts/blackList_whiteList.teal`

Recommendation

Stronger correctness checks where applicable will prevent an accidental misconfiguration and deployment. For example, we suggest that the round should be able to start, if spot and strike price are equal and with the following logic:

- `stp`
- `spp`
- `!=`
- `assert`

Reference
N/A

KS-AR-07 – CloseOut accepting transactions while the user still has funds

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	Medium	Easy

Description

The two contracts, `deposits.teal` and `withdraw.teal`, stored a reserve of Algos or ASA belonging to multiple users. They used local and global variables to determine at any given time the percentage of that reserve belonging to each user. The user may choose to call `CloseOut`, to opt out of the contract at any time. However, if the user still had Algos or ASA locked-in the contract, then this operation should fail. This was not the case, as `CloseOut` accepted all transactions, regardless of the user's local state.

Impact

If a user accidentally chooses to `CloseOut` while still having Algos or ASA locked-in the contract, they may be unable to access them anymore.

Evidence

```
274  handle_closeout:
275      // Handle CloseOut
276
277      // approve closeout
278      b finish
```

CloseOut in deposit.teal

Affected Resource

- `vaults/contracts/deposit.teal (Line 275)`
- `vaults/contracts/withdraw_round.teal (Line 263)`

Recommendation

Modify the `CloseOut` branch in the two contracts depending on the user's state. If the user has no pending deposit or if they have withdrawn their share, then the `CloseOut` should accept. Otherwise, `CloseOut` should reject the transaction.

Reference

N/A

KS-AR-08 – Group transactions initiated by the admin do not check the Sender of the second transaction

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Moderate

Description

Several times in this application, two transactions were being sent by the admin. One of them, the application call, was meant to trigger some part of the business logic. The `Sender` of this transaction was authenticated by the `admin_verify` subroutine. The other transaction was usually a pay or asset transfer transaction. With the exception of the `handle_premium` (in `vault.teal`), the `Sender` of this other transaction was never checked.

Impact

Because only the `Sender` of the application call is authenticated, the following scenario could happen. Let us consider the possibility of the administrator grouping by mistake their (authenticated) application call with another transaction not sent by them, but which is otherwise well-formed.

An attacker, wishing to exploit this possibility, will regularly send well-formed transactions (containing field values as expected by the smart contract), that also has in addition some other fields set, like `RekeyTo` or `CloseRemainderTo`. Most of the time, this transaction, by itself, will get rejected (as it is not part of a legitimate group) and nothing will happen. If, however, the administrator mistakenly groups their legitimate application call with this malicious transaction, this could have a high negative impact.

If both transactions get accepted, all their effects will apply, potentially including `RekeyTo` and `CloseRemainderTo`. We note that, for this to occur, the finding KS-AR-06 is also required to be in effect.

Evidence

See next page.

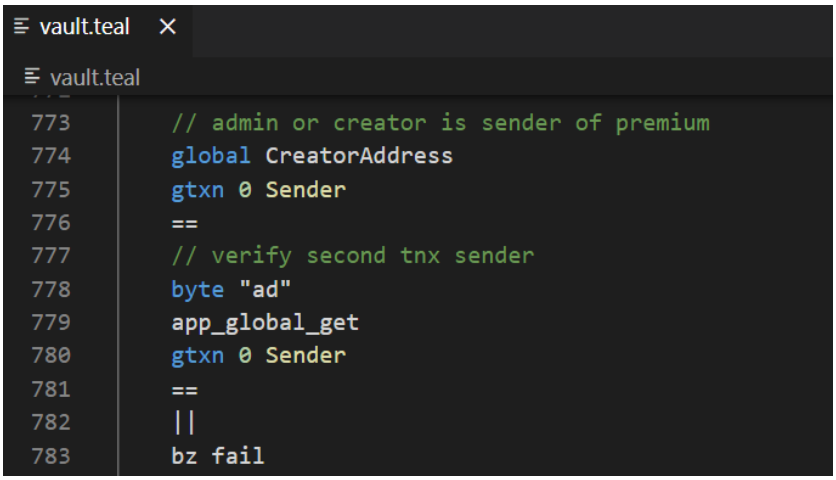
Affected Resource

- `vaults/contracts/vault.teal (Lines 429-482)`
- `vaults/contracts/vault.teal (Lines 873-907)`
- `vaults/contracts/vault.teal (Lines 1252-1281)`
- `vaults/contracts/vault.teal (Lines 2433-2454)`

Recommendation

Authenticating the `Sender` of both transactions would make the code overall more robust.

Where applicable, perform an additional check to authenticate the Sender of the second transaction, as performed in `handle_premium`:



`vaults/contracts/vault.teal`: correct check example

Reference

N/A

Evidence

vault.teal

```
1240    b finish
1241
1242  // END of SETTLEMENT PENDING
1243
1244  handle_settlement_complete:
1245      //****** Verifications ********//
1246      // 1. verify sender
1247      // 2. verify status
1248      // 3. verify txn details
1249      // 4. check round settlement s
1250
1251      // 1. verify admin is calling
1252      callsub admin_verify
1253
1254      // 2. this call should be avai
1255      byte "st"
1256      app_global_get
1257      int 3
1258      ==
1259      assert
1260
1261      // 3. atomic transfer verifications
1262      // verify group size
1263      // must be one txn
1264      global GroupSize
1265      int 2
1266      ==
1267      // verify that this txn is stateful contract call
1268      txn GroupIndex
1269      int 1
1270      ==
1271      &&
1272      // Algo transfer
1273      gtxn 0 TypeEnum
1274      int pay
1275      ==
1276      &&
1277      // App call
1278      gtxn 1 TypeEnum
1279      int appl
1280      ==
1281      &&
```

```
2525  admin_verify:
2526      byte "ad"
2527      app_global_get
2528      txn Sender
2529      ==
2530      global CreatorAddress
2531      txn Sender
2532      ==
2533      ||
2534      assert
2535
2536      retsub
```

Example of a group of transactions called by the administrator. The admin_verify subroutine will only verify the Sender of the current transaction (situated at index 0), but no Sender identity checks are performed on the pay transaction at index 0.

KS-AR-09 – Performing mathematical operations with very low amounts

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	High	Easy

Description

On occasion, the smart contracts performed a variation of the following operation:

```
user's_share * existing_value / sum_of all_user_shares
```

This was implemented using the `muldiv` subroutine. However, `muldiv` discarded the remainder of the division operation. In a similar note, users depositing 0 or a minute amount of assets may have unintended consequences.

Impact

- If a user could make very small deposits, an admin may be able to start a vault round due to existing pending deposit, but the amount of assets stored in the contracts may not be worth bidding on. In addition, if their share is "dwarfed" by other deposits, they may not have a share to withdraw, despite having participated in the vault.
- In the case of the currency being Algo, making a transfer of an amount less than 1000 Algos (to the user, or to the treasury/options settlement) may not be worth the transaction Fee. This may not hold true for ASAs.
- The sum of all shares of the users will not always equal the sum of all shares, but a tiny bit smaller (due to the discarded remainder).

Evidence

vault.teal

```
2611    // First B*C, A is deepest A/B/C
2612    muldiv:
2613        mulw                // multiply B*C. puts TWO u64s on stack
2614        int 0               // high word of C as a double-word
2615        dig 3               // pull C to TOS
2616        divmodw
2617        pop                 // pop unneeded remainder low word
2618        pop                 // pop unneeded remainder high word
2619        swap
2620        int 0
2621        ==
2622        assert              // ensure high word of quotient was 0
2623        swap                // bring C to surface
2624        pop                 // in order to get rid of it
2625        retsub
```

The `muldiv` subroutine. The remainder of the division gets discarded, which might lead to very small amounts accumulating in the vault.

Affected Resource

- `vaults/contracts/withdraw_round.teal`
- `vaults/contracts/vault.teal`
- `vaults/contracts/deposit.teal`

Recommendation

This might be mitigated first by ensuring that no deposits can be made, unless they are bigger than a suitable threshold. The same check should be performed when starting a vault round. Due to the users shares not adding up to their total due to rounding, there might be always some small amounts of currency staying in the vault contract. Over a very, very large number of rounds, these amounts may add up to very low, but potentially non-negligible amounts.

In addition, one option would be to avoid performing certain transactions if they are "too costly", but this largely depends on the intended business logic.

Reference

N/A

KS-AR-10 – Black/White list no duplicate check

Severity	**LOW**
Status	**RESOLVED**

Impact	Likelihood	Difficulty
Low	Low	Easy

Description

The black and white lists store a list of users according of the action they can or cannot perform. When adding an address to the lists, there was no check if the address already belonged to the list. This meant that the same address could be added multiple times in the same list without any check.

Impact

This could result in a lack of storage for both lists. Since both lists have a maximum limit of 62 addresses, this means that the absence of duplication check could result in having the same address saved multiple times and take the memory space of other addresses.

This would mean users would be prevented from being added to the whitelist, in the case of depositing while contract is paused, or the blacklist in the case of being banned from depositing funds.

Evidence


```
},
"addr_8": {
  "tb": "JKNZ7JHARUUKFFGQS3RUDP5SVUY66GK5PUZGQYGYFQJIZ5WT7BCDPS6F4I",
  "tt": 1
},
"addr_9": {
  "tb": "JKNZ7JHARUUKFFGQS3RUDP5SVUY66GK5PUZGQYGYFQJIZ5WT7BCDPS6F4I",
  "tt": 1
},
"addr_:": {
  "tb": "JKNZ7JHARUUKFFGQS3RUDP5SVUY66GK5PUZGQYGYFQJIZ5WT7BCDPS6F4I",
  "tt": 1
},
"addr_;": {
  "tb": "JKNZ7JHARUUKFFGQS3RUDP5SVUY66GK5PUZGQYGYFQJIZ5WT7BCDPS6F4I",
  "tt": 1
},
"addr_<": {
  "tb": "JKNZ7JHARUUKFFGQS3RUDP5SVUY66GK5PUZGQYGYFQJIZ5WT7BCDPS6F4I",
  "tt": 1
},
"addr_=": {
  "tb": "JKNZ7JHARUUKFFGQS3RUDP5SVUY66GK5PUZGQYGYFQJIZ5WT7BCDPS6F4I",
  "tt": 1
},
```


Black/White list duplicated address

Affected Resource

- `vaults/contracts/blackList_whiteList.teal (Lines 98-136)`

Recommendation

Check duplicates before adding a new address to the list (for instance, looping over all current addresses and check against the new one).

Reference

N/A

KS-AR-11 – TWAP value error management

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Easy

Description

In the price-feeder, only median values greater than zero were considered when computing the TWAP (time-weighted average price) value.

In the cases that feeders have not reported any prices, or that prices have dropped to zero, the TWAP computation would end by a division by zero.

Impact

When the above conditions are met, all TWAP requests will systematically fail, leading to a misinterpretation of the state of the system (unexpected error vs. zero value).
In the case of prices dropped to zero (considering that value is allowable), such a value will never be returned, thus introducing a bias into the system.

Evidence

```
232
233             twap.store(sum.load() / divider.load()),
234
```

medianizer_contract.py – demonstrating where a potential division by zero might occur.

Affected Resource

- `price-feeder/deploy/src/contracts/medianizer_contract.py (Line 233)`

Recommendation

Check that the divider is not equal to zero before performing division.

Reference

N/A

KS-AR-12 – Incorrect int to ASCII conversion in blackList_whiteList

Severity	INFORMATIONAL
Status	RESOLVED

Impact	Likelihood	Difficulty
-	-	-

Description

The `blackList_whiteList.teal` implemented a "guardrails" functionality for the decentralized vault. The two contracts stored a list of user addresses. Blacklisted users were not allowed to deposit, while whitelisted users could make a deposit while the contract is paused. During our analysis we discovered that the code to generate the labels for the addresses was not implemented correctly for indices beyond 10.

Due to the incorrect int to ASCII incorrect conversion, after the first 9 blacklisted/whitelisted addresses are stored, subsequent addresses would be stored in the contract (using labels such as "`addr_;`" depending on the ASCII table).

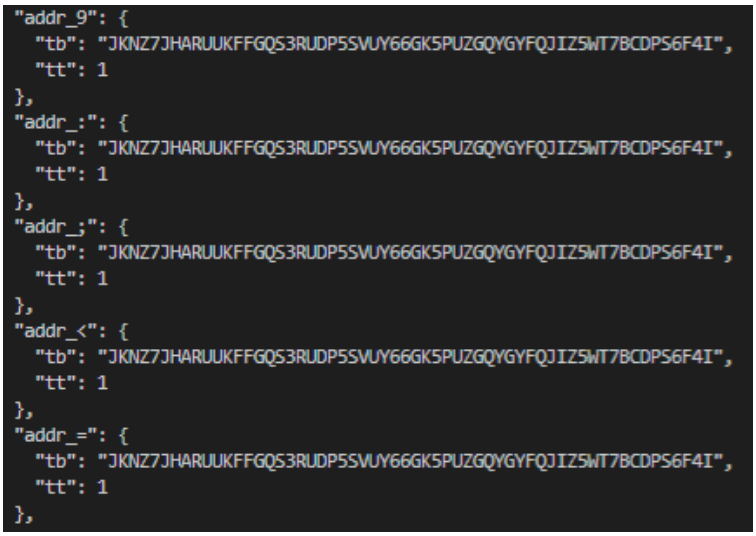
Impact

Due to the mismatch between the expected code functionality and the actual code functionality might cause unexpected behavior, particularly when combined with other smart contracts. This might slow down future code development and maintenance.

Evidence

```
118     // create index
119     byte "0"
120     int 0
121     byte "AddrCount"
122     app_global_get
123     int 1
124     +
125     int 48
126     +
127     setbyte
```

Int to ascii conversion, managing only the first digit



After addr_9, addresses are not written correctly

Affected Resource

- `vaults/contracts/blackList_whiteList.teal (Lines 118-127)`

Recommendation

Complete the int to ASCII conversion for indexes > 10, adding one digit for each power of 10.

Reference

N/A

KS-AR-13 – Mismatches between the documentation, comments and the code

Severity	INFORMATIONAL

Impact	Likelihood	Difficulty
-	-	-

Description

On a few occasions we noticed a slight mismatch between either the code and the documentation, as well as the code and the surrounding comments. Whenever this was the case, we have considered the code as the "source of truth".

Impact

Where there are discrepancies between the specification and implementation, future maintenance, development, and review activities could be slowed down.

Evidence

In the following example, the comment “if not algo fail” contradicts the code, which should fail in case the currency is Algo.

```
311  handle_init:
312      // atomic transfer called by creator: Algo transfer and app call
313      // this contract to optin to currency, if it's ASA
314
315      // remember currency
316      byte "dc"
317      app_global_get
318      store 0
319
320      // if not algo fail
321      load 0
322      int 0
323      >
324      assert
```

Screen caption that illustrates a code-comment contradiction. In this case, the comment states that the function will fail if it is not algo. In practice, the code will stop executing in the opposite case, if the currency is Algo (as the assert will fail).

Affected Resource

- `vault.teal` & `deposit.teal` - mismatch code-comment (nr params).
- `vault.teal` - `handle_init` code directly contradicts the comment.
- `medianizer_contract.py` & `methods.py` vs. Tellor Reporter Golang App.pdf.

Recommendation

Ensure better consistency between documentation, code, and tests.

Reference

N/A

KS-AR-14 – Code duplication

Severity	INFORMATIONAL
Status	RESOLVED

Impact	Likelihood	Difficulty
-	-	-

Description

Some functionality shared between contracts is duplicated across all of them.

Impact

Duplicated code is used as a code maturity metric in the industry to point out how maintainable the code base is. It is also a possible entry point for new bugs as code duplication leads to mistakes when updating/rewriting the codebase.

Evidence

```
81  @Subroutine(TealType.bytes)
82  def convert_uint_to_bytes(arg):
83      string = ScratchVar(TealType.bytes)
84      num = ScratchVar(TealType.uint64)
85      digit = ScratchVar(TealType.uint64)
86
87      return Seq([
88              string.store(Bytes("")),
89              For(num.store(arg), num.load() > Int(0), num.store(num.load() / Int(10))).Do(
90                  Seq([
91                      digit.store(num.load() % Int(10)),
92                      string.store(
93                          Concat(
94                              Substring(
95                                  Bytes('0123456789'),
96                                  digit.load(),
97                                  digit.load() + Int(1)
98                              ),
99                              string.load()
100                         )
101                     )
102                 ])
103
104             ),
105             string.load()
106         ])
```

methods.py - first implementation

```
66      @Subroutine(TealType.bytes)
67      def convert_uint_to_bytes(arg):
68          string = ScratchVar(TealType.bytes)
69          num = ScratchVar(TealType.uint64)
70          digit = ScratchVar(TealType.uint64)
71
72          return Seq([
73                  string.store(Bytes("")),
74                  For(num.store(arg), num.load() > Int(0), num.store(num.load() / Int(10))).Do(
75                      Seq([
76                          digit.store(num.load() % Int(10)),
77                          string.store(
78                              Concat(
79                                  Substring(
80                                      Bytes('0123456789'),
81                                      digit.load(),
82                                      digit.load() + Int(1)
83                                  ),
84                                  string.load()
85                              )
86                          )
87                      ])
88
89                  ),
90                  string.load()
91              ])
```

medianizer_contract.py – Same implementation

Affected Resource

- change_governance
  - price-feeder/deploy/src/contracts/methods.py (Lines 109-126)
  - price-feeder/deploy/src/contracts/medianizer_contract.py (Lines 147-164)
- convert_uint_to_bytes
  - price-feeder/deploy/src/contracts/methods.py (Lines 81-106)
  - price-feeder/deploy/src/contracts/medianizer_contract.py (Lines 66-91)

Recommendation

Refactor the code such that functionalities shared in multiple contracts is found in only one file.

Reference

N/A

KS-AR-15 – Inconsistent naming conventions

Severity	INFORMATIONAL

Impact	Likelihood	Difficulty
-	-	-

Description

On occasion, the naming conventions for variable names, functions etc. changes from one part of the code base to another.

Impact

Using inconsistent name variables can lead to errors during the development phase and can make it more difficult to create static analysis rules for the code database.

Evidence

```
63      txna ApplicationArgs 0
64      byte "admin_change"
65      ==
66      &&
67      bnz handle_admin_change
68
69      // ###################
70      // Add address
71      // ###################
72      // two param: method name and address to be saved
73      txn NumAppArgs
74      int 2
75      ==
76      txna ApplicationArgs 0
77      byte "add-addr"
78      ==
79      &&
80      bnz handle_add_address
```

blackList_whiteList.teal: "-" as separator for only one method

```
376     return Cond(
377         [contract_method == Bytes("change_governance"), change_governance()],
378         [contract_method == Bytes("change_medianizer"), change_medianizer()],
379         [contract_method == Bytes("tip"), tip()],
380         [contract_method == Bytes("report"), report()],
381         [contract_method == Bytes("slash_reporter"), slash_reporter()],
382         [contract_method == Bytes("add-addr"), add_addr()],
383         [contract_method == Bytes("del-addr"), del_addr()],
384     )
```

methods.py: "-" as separator for only two methods

```
453         contract_method = Txn.application_args[0]
454         return Cond(
455             [contract_method == Bytes("activate_contract"), activate_contract()],
456             [contract_method == Bytes("change_governance"), change_governance()],
457             [contract_method == Bytes("get_values"), get_values()],
458             [contract_method == Bytes("read-price"), read_price()],
459             [contract_method == Bytes("get-twap"), get_twap()],
460         )
```

medianizer_contract.py: "-" as separator for only two methods

Affected Resource

- `vaults/contracts/blackList_whiteList.teal (Lines 77)`
- `price-feeder/deploy/src/contracts/methods.py (Lines 382-383)`
- `price-feeder/deploy/src/contracts/medianizer_contract.py (Lines 458-459)`

Recommendation

Where applicable, pick one naming convention and apply it consistently e.g., using "`handle_`" as a prefix for handling incoming transactions, or always using _ when separating words in variable names.

Reference

N/A

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase the team works through the following categories:

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

These categories incorporate common vulnerabilities such as the OWASP Top 10

Tools

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

- Visual Studio Code
- Algorand sandbox + Project JavaScript tests
- Algorand CLI: GOAL

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system

Impact
The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

> **High:**
> The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client
>
> **Medium:**
> It is reasonable to assume that the vulnerability would have a measurable affect on the company and systems that may cause minor financial or reputational damage.
>
> **Low:**
> There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood
The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

> **High:**
> It is extremely likely that this vulnerability will be discovered and abused
>
> **Medium:**
> It is likely that this vulnerability will be discovered and abused by a skilled attacker
>
> **Low:**
> It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty
Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

> **Easy:**
> The vulnerability is easy to exploit or has readily available techniques for exploit
>
> **Moderate:**
> The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.
>
> **Difficult:**
> The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Adina Nedelcu	**Security Engineer**	**adina.nedelcu@kudelskisecurity.com**
Maxime Buser	**Security Engineer**	**maxime.buser@kudelskisecurity.com**
Ronan Le Gallic	**Lead Engineer**	**ronan.legallic@nagra.com**