SMART CONTRACT AUDIT PUBLIC REPORT

Audit of Algogard Smart Contract

VPQ-20220023

Algogard

23rd March 2022

VANTAGEPOINT

Security at the Speed of Development

TABLE OF CONTENTS

1. Executive Summary 3

Overview 3

Vulnerability Overview 3

2. Project Details 5

Scope 5

Version History 6

3. Risk Assessment 7

Overview of Components and Their Vulnerabilities 7

1. Algogard Smart Contract Audit 7

4. Detailed Description of Vulnerabilities 9

2. Algogard Smart Contract Audit 9

1.1. Incorrect Enforcement of Fees Paid to Treasury During Liquidation 9

1.2. Incorrect Calculation Resulting in Zero Payout to Managers and Founders 11

1.3. Erroneous Subroutine Stops Liquidation From Execution After 46 Minutes 13

1.4. Insecure Storage of Mnemonic Seed Phrases 16

1.5. Incorrect Application Logic Limits Number of External Application Opt-In 17

1.6. Incorrect Assert Logic 19

1.7. Incorrect Value Set for Max Supply of DAO Token 20

1.8. Incorrectly Implemented For Loop Operations 22

1.9. Missing Algorand Standard Asset ID Checks during ASA Transfer 24

1.10. Insufficient On-Chain Validation Against Altered Transactions 27

1.11. Lack of Checks for Locking Vote App 38

1.12. Redundant Code 39

1.13. Price Oracle and DAO Manager 41

1.14. Incorrect Comments 43

5. Appendix 45

Disclaimer 45

Scope of Audit 45

1. EXECUTIVE SUMMARY

OVERVIEW

Vantage Point Security Pte Ltd was engaged by Tapera Inc. to conduct an independent security assessment of the smart contracts comprising the Algogard Gard suite of services. The intent of the review was to identify security vulnerabilities, weaknesses and any instances of non-compliance to best practices or regulatory requirements. Testing commenced on the 3rd March 2022 and was completed on the 23rd March 2022.

Algorand smart contract security review was conducted based on the following items provided for audit.

- PyTeal Code
    - Private Repo
        - https://github.com/Tapera-Finance/CodeAudit/tree/WholeAudit
          Commit ID 07265bb545cc475ea29e485c2575e5ec220075f0
- Documents
    - Algogard Whitepaper
        - https://www.algogard.com/white-paper.pdf

Vantage Point performed this review by first understanding the high-level business logic of Gards functionality and the interactions between different smart contracts within the provided documentation. We sought clarifications on potential issues, discrepancies, and flaws throughout the review with the Algogard team.

Throughout the smart contract audit, Vantage Point had active communication with the GARD team and received timely and helpful support. Provided documentation had details of the business logic and definitions of expected transaction groups for each user flows, making sure the code is well documented and up-to-date.

An audit was conducted on the provided PyTeal code to identify any weaknesses, vulnerabilities, and non-compliance to Algorand best practices. Test cases included in this review have been amended in the appendix of this document for completeness.

The following noteworthy issues were identified during this review.

1. **Incorrect Enforcement of Fees Paid to Treasury During Liquidation**
2. **Missing Algorand Standard Asset ID Checks during ASA Transfer**
3. **Incorrect Calculation Resulting in Zero Pay-out to Managers and Founders**

The outcome of this Algorand Smart Contract Security Review engagement is provided as a detailed technical report that provides the Smart Contract owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the identified technical issue.

VULNERABILITY OVERVIEW

Severity	Count	Open	Closed
Critical	**0**	**0**	**0**
High	**2**	**0**	**2**
Medium	**6**	**0**	**6**
Low	**3**	**3**	**0**
Observational	**3**	**2**	**1**
Summary	**14**	**5**	**9**

Vulnerability Risk Score

All vulnerabilities found by Vantage Point will receive and individual risk rating based on the following four categories.

CRITICAL COMPONENT RISK SCORE

Critical severity findings relate to an issue, which requires immediate attention and should be given the highest priority by the business as it will critically impact business interest critically.

HIGH COMPONENT RISK SCORE

HIGH severity findings relate to an issue, which requires immediate attention and should be given the highest priority by the business.

MEDIUM COMPONENT RISK SCORE

A MEDIUM severity finding relates to an issue, which has the potential to present a serious risk to the business.

LOW COMPONENT RISK SCORE

LOW severity findings contradict security best practice and have minimal impact on the project or business.

OBSERVATIONAL

Observational findings relate primarily to non-compliance issues, security best practices or are considered an additional security feature that would increase the security stance of the environment which could be considered in the future versions of smart contract.

2. Project Details

Scope

Contact Name	David McCabe
Application Name	Algogard
Testing Period	3rd March 2022 – 23rd March 2022
GIT Commit ID	07265bb545cc475ea29e485c2575e5ec220075f0
Items Completed	Vantage Point completed the agreed Security assessment for below items. • cdp_escrow.py • price_validator.py • reserve_logic.py • Stake.py • treasury.py • utils.py • Vote_fee.py • Vote_lib.py • Vote_manager.py

Component	Review Type	Status
Algorand Smart Contract	Smart Contract Security Review	Completed
Algorand Smart Contract	Smart Contract Security Review Retest	Not Completed

Version History

Date	Version	Release Name
23rd March 2022	v0.1	Draft
24th March 2022	v0.2	QA Release
24th March 2022	V1.0	Final
29th March 2022	V1.1	Retest Updates
30th March 2022	V1.2	QA Release 2

3. Risk Assessment

This chapter contains an overview of the vulnerabilities discovered during the project. The vulnerabilities are sorted based on the risk categories of CRITICAL, HIGH, MEDIUM and LOW. The category OBSERVATIONAL refers to vulnerabilities that have no risk score and therefore have no immediate impact on the system.

Overview of Components and Their Vulnerabilities

1. Algogard Smart Contract Audit		HIGH RISK
1.1. Incorrect Enforcement of Fees Paid to Treasury During Liquidation	Closed	HIGH RISK
1.2. Incorrect Calculation Resulting in Zero Pay-out to Managers and Founders	Closed	HIGH RISK
1.3. Erroneous Subroutine Stops Liquidation from Execution After 46 Minutes	Closed	MEDIUM RISK
1.4. Insecure Storage of Mnemonic Seed Phrases	Closed	MEDIUM RISK
1.5. Incorrect Application Logic Limits Number of External Application Opt-In	Closed	MEDIUM RISK
1.6. Incorrect Assert Logic	Closed	MEDIUM RISK
1.7. Incorrect Value Set for Max Supply of DAO Token	Closed	MEDIUM RISK
1.8. Incorrectly Implemented For Loop Operations	Closed	MEDIUM RISK
1.9. Missing Algorand Standard Asset ID Checks during ASA Transfer	Open	LOW RISK

1.10. Insufficient On-Chain Validation Against Altered Transactions	Open	LOW RISK	
1.11. Lack of Checks for Locking Vote App	Open	LOW RISK	
1.12. Redundant Code	Open	OBSERVATIONAL	
1.13. Price Oracle and DAO Manager	Open	OBSERVATIONAL	
1.14. Incorrect Comments	Closed	OBSERVATIONAL	

4. DETAILED DESCRIPTION OF VULNERABILITIES

2. Algogard Smart Contract Audit

HIGH RISK

1.1. Incorrect Enforcement of Fees Paid to Treasury During Liquidation

HIGH RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Incorrect logic or inconsistency in the calculation of values which are used in a Transfer or Asset Transfer transaction may result in financial loss to the parties involved in the transaction. It is important to ensure the calculation logic for such values are carried out in a correct and consistent manner to ensure accurate and consistent values are referred when transactions are approved.

DESCRIPTION

Instance 1

Affected File

- priceValidator.py
  - liquidate - Line 87-109

Based on the GARD Whitepaper page 8-9, during liquidation, remaining GARD balance after paying the GARD_DEBT of the CDP escrow to the reserve, should be distributed between the CDP escrow's original owner and the devfee address (Treasury).

```
After the GARD debt has been repaid to the reserve, 20% of the remainingGARD is
sent back to the reserve, while the rest of the collateral (80%) is returned to
the account that was liquidated (the CDP holder).
```

For example, if GARD left after paying the GARD_DEBT is 10 GARDs, this should be distributed per below.

- 8.000000 GARD to CDP's original owner (80%)
- 2.000000 GARD to Treasury (20%)

However, the logic implemented enforces an incorrect amount for the fees paid to Treasury.

As Gtxn[3].asset_amount which represents the devfee being paid only needs to be greater than or equal to Gtxn[4].amount()/Int(5), for the same scenario where 10 GARDs needs to be distributed, it would be distributed per below.

- 8.333333 GARD to CDP's original owner
- 1.666667 GARD to Treasury

In cases where the liquidator is the owner (self-liquidation) of the CDP contract account, the smart contract incentivizes the liquidator to transfer less to the Treasury and more to the CDP owner.

RECOMMENDATION

Review the whitepaper and the code and check the correct amount to be enforced through Teal logic validation so both the documentation and the code are in sync.

REGRESSION TESTING COMMENTS

26th March 2022 – This issue is closed.

Based on the commit abee2af8163c2e4be888c7cefc59e23cc368cb2c, incorrect logic for enforcing amount of fee paid to Treasury during liquidation has been fixed.

price_validator.py – Line 109

```
Gtxn[3].asset_amount() == Gtxn[4].asset_amount()/Int(4),
```

VULNERABILITY REFERENCES

PyTeal Documentation – Arithmetic Operations

https://pyteal.readthedocs.io/en/latest/arithmetic_expression.html

1.2. Incorrect Calculation Resulting in Zero Payout to Managers and Founders

HIGH RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Incorrect logic or inconsistency in calculation of values which are used in Transfer or Asset Transfer transactions may result in financial loss to the parties involved in the transaction. It is important to ensure the calculation logic for such values are carried out in a correct and consistent manner to ensure accurate and consistent values are referred when transactions are approved.

DESCRIPTION

Affected File/Code

- treasury.py
  - payout - Line 68-105

The Algogard whitepaper page 8 states the following.

```
The Treasury pays 18% of its fee proceeds (in ALGO) to the DAO Manager account
each quarter. It also pays 2% of its fee proceeds to a founder account each
quarter.
```

In treasury.py, incorrect calculation logic for pay-out values for manager and founder were observed. In line 77, the value of treasury balance was put into a global state **ALGO_BALANCE.** Afterwards, the global state **ALGO_BALANCE** was used to deduct from the value of the treasury balance in line 78. Assuming value of treasury balance is 1000 ALGO, the following calculation will take place: ***(1000 - 1000) * Manager's Percentage Cut / 100***, which results in 0. As the same calculation logic was used to calculate the founder's pay-out value, this would effectively impact both manager and founder as they would receive 0 pay-outs every quarter.

RECOMMENDATION

Review the calculation logic to ensure that the arithmetic operation is implemented correctly as intended, as well as how the parameters used in the calculation are being passed into the arithmetic operation and how the result is being stored.

Following incorrect logic for global state modification can be removed.

```
App.globalPut(Bytes("ALGO_BALANCE"), Balance(Int(1))),
```

REGRESSION TESTING COMMENTS

28th March 2022 – This issue is closed.

Instance 1

Based on the commit 05faf8a4246e6be5e22557be87335886b2855dc, incorrect use of global state "ALGO_BALANCE" has been remediated and the logic correctly calculates the pay-out amount for the founder and the manager.

VULNERABILITY REFERENCES

PyTeal Documentation – Arithmetic Operations:

https://pyteal.readthedocs.io/en/latest/arithmetic_expression.html

1.3. Erroneous Subroutine Stops Liquidation From Execution After 46 Minutes

MEDIUM RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

When a CDP's collateral value drops below 115% of the GARD minted, CDP is liquidated through a Dutch auction where the price of the collateral in GARD drops from 115% of the GARDs minted to 100% in 6 minutes. However, due to lack of checks before triggering the subroutine auction_price(), auction_price() is executed even after 6 minutes since the start of auction and eventually, after 2761 seconds from the start of auction, the auction_price() arrives at a negative value and returns an error. As this subroutine is called regardless of the time difference between current block's timestamp and the start of auction, once it has been 2761 seconds from the start of an auction for a CDP, all liquidation will fail for the specific CDP.

DESCRIPTION

Instance 1

Affected Files

- price_validator.py
    - liquidate - Line 87-109
    - auction_price() - Line 21-33

It was noted that during liquidation, the smart contract logic makes use of subroutine auction_price() to retrieve the dutch auction price of the CDP which linearly decreases from 115% of the collateral value to 100%.

As below part of the logic is used to decrease the auction price, Subroutine **auction_price()** has following trends.

```
(Global.latest_timestamp() - App.localGet(Txn.sender(),
Bytes("UNIX_START")))/Int(24))
```

1. Upon start of auction where (Global.latest_timestamp() - App.localGet(Txn.sender(), Bytes("UNIX_START")) == 0

- $auction\_price() = 1.15 * GARD\_DEBT$

2. Within 360 seconds from the start of auction where 0 < (Global.latest_timestamp() - App.localGet(Txn.sender(), Bytes("UNIX_START")) <=360

- $1.15 * GARD\_DEBT > auction\_price() \geq 1 * GARD\_DEBT$

3. From 361 seconds to 2760 seconds from the start of the auction (In case the liquidation was not successful) where 361<(Global.latest_timestamp()-App.localGet(Txn.sender(),Bytes("UNIX_START")) <=2760

- $1 * GARD\_DEBT > auction\_price() \geq 0$

4. After 2760 seconds from the start of the auction

- Returns error as $auction\_price()$ becomes negative and only unsigned integer is allowed

Within the liquidate logic, subroutine auction_price() is always called even when it has been more than 6 minutes from the start of the auction.

price_validator.py - liquidate

```
Gtxn[2].asset_amount() + Gtxn[3].asset_amount() + Gtxn[4].asset_amount() >=
Max(App.localGet(Txn.sender(), Bytes("GARD_DEBT")), auction_price()),
```

Under following conditions, auction_price() subroutine would always fail and therefore, "liquidate" transaction group would always be rejected.

- CDP's auction is started
- CDP's auction was not successful for more than 2760 seconds since the start of the auction
  - Community participants, keepers and DAO Manager failed in successfully liquidating the CDP for 46 minutes

Instance 2

Affected Files

- price_validator.py
  - liquidate - Line 87-109
  - auction_price() - Line 21-33

This instance has been self-reported and remediated by the Algogard team during regression testing. It was noted that the subroutine auction_price() makes use of wrong denominator when calculating the rate of deduction based on the number of seconds from **UNIX_START**. Below code snippet calculates the equation for the auction price per equation below.

price_validator.py - auction_price

```
temp.store(temp.load()-(App.localGet(Txn.sender(),
Bytes("GARD_DEBT"))*(Global.latest_timestamp() - App.localGet(Txn.sender(),
Bytes("UNIX_START")))/Int(24)))
```

$$temp = GARD_{DEBT} - GARD_{DEBT} * \frac{\Delta t}{24}$$

Where $\Delta t = t_{latest\_timestamp} - t_{auction\_start}$

Therefore, 1 second after the auction start, the logic would update the price to below.

$$temp = GARD_{DEBT} - GARD_{DEBT} * \frac{1}{24} = \frac{23}{24} GARD_{DEBT} \approx 0.9583\, GARD_{DEBT}$$

Instead of taking 6 minutes or 360 seconds to bring the price down from 1.15 GARD_DEBT (115%) to 1.00 GARD_DEBT (100%), it would drop the **auction_price()** by around 4.16% every second, which drops the price to 100% 4 seconds after the auction start.

RECOMMENDATION

Instance 1

- Add checks within liquidate to make sure subroutine auction_price() is not triggered when auction_price() is expected to return an error
- Add validations within subroutine auction_price() to handle scenarios which would trigger an error based on current logic so that transaction group liquidate can still be approved

Instance 2

- As the deduction is in %, denominator of the logic for calculating the price deduction should be further divided by 100

REGRESSION TESTING COMMENT

28th March 2022 – This issue is closed.

Instance 1 & 2

Based on commit IDs noted below, both issues highlighted under instance 1 and 2 have been remediated to ensure correct decrement of auction_price() during auction and prevention of erroneous subroutine calls after 2760 seconds.

- e877442d2937e9b00db341ffc22f52685597d110
- c05faf8a4246e6be5e22557be87335886b2855dc

VULNERABILITY REFERENCES

PyTeal Documentation – Arithmetic Operations:

https://pyteal.readthedocs.io/en/latest/arithmetic_expression.html

1.4. Insecure Storage of Mnemonic Seed Phrases

MEDIUM RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

A mnemonic seed phrase is a series of words generated by cryptocurrency wallets that provide access to the crypto associated with that wallet. During development of smart contracts, developers may hard code such phrases in the code for ease of access and forget about removing them in post-deployment of contracts. Due to the open-source nature of smart contracts, malicious actors may easily gain hold of the phrases and obtain access to the wallets that the phrases are associated with, and the impact would be potentially severe if the wallet contains funds in the MainNet as well.

DESCRIPTION

Instance 1 – Vote_manager.py – Line 14

Affected File/Code

- Vote_manager.py
    - Line 14

It was noted that the affected file included mnemonic phrases as comments.

RECOMMENDATION

1. Create .gitignore file to specify files that should not be tracked and add .env file to the list.
2. Setup a .env file to store the mnemonic phrases.
3. Retrieve the phrases from .env to contract for testing purposes.

REGRESSION TESTING COMMENT

28th March 2022 – This issue is closed.

VULNERABILITY REFERENCES

Consensys – Prevent Making Your Secrets Public:

https://consensys.net/blog/developers/how-to-avoid-uploading-your-private-key-to-github-approaches-to-prevent-making-your-secrets-public/

CWE:
https://cwe.mitre.org/data/definitions/922.html

OWASP:
https://www.owasp.org/index.php/Mobile_Top_10_2014-M2

1.5. Incorrect Application Logic Limits Number of External Application Opt-In

MEDIUM RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Smart contracts use local states, global states and logic to ensure certain transactions can be limited to a specific condition based on the applicable business logic. However, if the logic used to limit the approval of limited transactions has flaws, unexpected transactions could be approved or cause a stand-still to the smart contract's functionality.

DESCRIPTION

Instance 1

Affected File

- price_validator.py
    - app_check - Line 200-211

This issue was reported by the Algogard Dev Team during the review. Subsequent commit made was reviewed by Vantage Point to validate if the commit made sufficiently addressed the identified issue.

It was noted that the application makes use of app_check to verify the transaction details of below transaction group.

- Voting from CDP with Arbitrary Application Call (Limit opt-in up to 3 applications)
    - Gtxn[0]
        - Payment of "account_id" microAlgos
        - From userAddress
        - to userAddress
    - Gtxn[1]
        - Application Call
        - From CDP
        - AppID Arbitrary App ID other than Validator App ID
    - Gtxn[2]
        - Application Call
        - From CDP
        - AppID priceValidator
        - App.Args[0]=="AppCheck"

The logic rejects the transaction group if the local state **External_APPCOUNT** is equal to or greater than 3 with below line of code.

```
App.localGet(Txn.sender(), Bytes("EXTERNAL_APPCOUNT")) < ex_apps_limit,
```

For every successful application call, the value of local state **External_APPCOUNT** increases by 1 with below code.

```
App.localPut(Txn.sender(), Bytes("EXTERNAL_APPCOUNT"), App.localGet(Txn.sender(),
Bytes("EXTERNAL_APPCOUNT")) + Int(1)),
```

However, even when the application call is intended to clear state or close out, the local state **EXTERNAL_APPCOUNT** value increases regardless and therefore, do not allow further close out or clear state application calls afterwards. This would mean that only 3 application calls can be made from CDP to arbitrary app ID other than priceValidator contract, regardless of the Txn.on_completion() value.

RECOMMENDATION

Instance 1

Implement a logic where the value of *ex_apps_limit* decreases when there are "clearing" application calls such as OnComplete.CloseOut and OnComplete.ClearState.

REGRESSION TESTING COMMENT

15th March 2022 - This issue is closed.

Instance 1

Commit e4004e27c5e6566ef1ac3f62e56cb62af1925f5d was made to add a logic for increasing and decreasing the value of **ex_apps_limit** local state depending on the application call type to prevent lock-out and enforcing the limit in number of applications the cdp_escrow can opt-in to.

VULNERABILITY REFERENCES

Algorand Developer Portal – Application Call Transaction:

https://developer.algorand.org/docs/get-details/transactions/transactions/#application-call-transaction

1.6. Incorrect Assert Logic

MEDIUM RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Smart contracts use assert checks to ensure transactions fulfil required conditions first before the transactions can get approved. However, if the assert check used to validate transactions has flaws, unexpected transactions could be approved or cause a stand-still to the smart contract functionality.

DESCRIPTION

Instance 1:

Affected File

- Stake.py
    - unstake - Line 141-159

The incorrect assert logic at line 148 prevents users from unstaking an amount lower than their current staked amount:

Stake.py – unstake – Line 148

```
current_stake(App.id(), sender) <= amount,
```

RECOMMENDATION

Instance 1

Change the comparison operator at Line 148 from '<=' to '>=' so users can unstake an amount lower than their current total staked amount.

REGRESSION TESTING COMMENT

28th March 2022 – This issue is closed.

Based on the commit 061286798ff24ef39d39a628cc2a80233a3b21a6, incorrect assert logic for unstake has been remediated.

VULNERABILITY REFERENCES

PyTeal Documentation – Checking Conditions Assert:

https://pyteal.readthedocs.io/en/stable/control_structures.html?#checking-conditions-assert

1.7. Incorrect Value Set for Max Supply of DAO Token

MEDIUM RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

During issuance of Algorand Standard Assets (ASAs), attributes such as total supply, decimals, manager address, reserve address, freeze address and clawback address should be set according to requirements.

DESCRIPTION

Affected File

- Vote_fee.py - Line 10

The Algogard whitepaper page 5 states the following.

```
GAIN has a fixed total supply of 2 billion tokens, out of which there are currently 0 in circulation. 1.04B are slated for the public while the remaining 960M are to be used for private fundraising, hiring advisors, and compensating the team.
```

It was observed in the following file that the value of total supply for GAIN was incorrectly set to 200 million instead of 2 billion tokens which was not in accordance with what was documented in the whitepaper provided:

Vote_fee.py - Line 10

```
# Constants
ASSET_TOTAL = Int(200000000)
VOTE_INTERVAL = Int(23)
VOTE_LENGTH = Int(24)
MIN_VAL = 0
MAX_VAL = 30
STARTING_RESULT = Int(20)
```

RECOMMENDATION

Use the E notation to specify numbers that are too large or small to be conveniently written in decimal form to improve readability and reducing the possibility of human error.

Example:

```
initial_supply = Int(int(2e9))
```

REGRESSION TESTING COMMENT

28th March 2022

– This issue is closed.

Based on the commit ID 061286798ff24ef39d39a628cc2a80233a3b21a6, the value of ASSET_TOTAL has been updated to 2 billion tokens with 6 decimals.

Vote_fee.py – Line 10

```
ASSET_TOTAL = Int(2000000000000000)
```

VULNERABILITY REFERENCES

Algorand Developer Portal – Algorand Standard Assets (ASAs)

https://developer.algorand.org/docs/get-details/asa/?from_query=ASA#immutable-asset-parameters

1.8. Incorrectly Implemented For Loop Operations

MEDIUM RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

For-loops are used to perform iterative operations based on the termination condition. If the conditions are set wrongly, incorrect loop operation could result in unexpected results.

DESCRIPTION

Instance 1

Affected File

- Stake.py
  - check_all_votes – Line 20-32

It was noted that the for-loop implemented in subroutine **check_all_votes** had an incorrect termination condition set, causing an extra loop to be executed.

Stake.py – check_all_votes

```
<REDACTED
For(i.store(Int(0)), i.load() <= current_votes, i.store(i.load() + Int(1))).Do(
<REDACTED>
```

In a scenario where `App.globalGet(Bytes("Num_votes"))=1`, the actual number of loops executed by the logic would be 2, as the termination condition is set to be `i.load() <= current_votes` instead of `i.load() < current_votes`. The logic executes twice when `i=0` and `i=1` and exits before the 3rd iteration.

Instance 2

Affected File

- Vote_fee.py
  - close_vote – Line 132-139

It was noted that the smart contract logic implemented within the for loop unintentionally skips the first index 0 and thus does not allow the choice with index 0 to be the "winner" when a vote closes.

Vote_fee.py – close_vote

```
<REDACTED>
For(i.store(Int(1)), i.load() <= max_val, i.store(i.load() + Int(1))).Do(
<REDACTED>
```

This could exclude vote option with index 0 from becoming the winner during close_vote operation.

RECOMMENDATION

Use correct termination condition and start condition for iterative operations using for loop to ensure unnecessary loops are not executed and thus return an unexpected value.

Instance 1

Update the termination condition of the for loop from <= to < per below example.

```
i.load() < current_votes
```

Instance 2

Update the start condition of the for loop from i.store(Int(1)) to i.store(Int(0)) per below example.

```
i.store(Int(0)), i.load()
```

REGRESSION TESTING COMMENT

29th March 2022 – This issue is closed.

Instance 1

Based on the commit c48ddf6811c7248192adb505fee7c8dae84ec37f, the termination condition of the for loop has been remediated per recommendation.

Stake.py – Line 26

```
For(i.store(Int(0)), i.load() < current_votes, i.store(i.load() + Int(1))).Do(
```

Instance 2

Based on the commit c48ddf6811c7248192adb505fee7c8dae84ec37f, start condition of the affected for loop has been remediated per recommendation.

Vote_fee.py – Line 132

```
For(i.store(Int(0)), i.load() <= max_val, i.store(i.load() + Int(1))).Do(
```

VULNERABILITY REFERENCES

PyTeal – For

https://pyteal.readthedocs.io/en/stable/api.html?highlight=for#pyteal.For

1.9. Missing Algorand Standard Asset ID Checks during ASA Transfer

LOW RISK

VULNERABILITY TRACKING

STATUS: **Open**

BACKGROUND

In Algorand smart contracts, Algorand Standard Assets (ASA) can be transferred through asset transfer transactions. During such transactions, it is necessary to validate the asset ID of the ASA being transferred to ensure the correct type of ASA is being transferred.

DESCRIPTION

Instance 1 – Mint More GARD

Affected File/Code

- reserve_logic.py
  - more_gard – Line 95-105
- price_validator.py
  - more_gard – Line 191-213

It was noted that during the **more_gard** operation, the smart contract lacked checks to ensure ASA being transferred is of a correct ASA ID, which should be equal to **Int(stable_id)**.

The Gtxn[2] of the transaction group for **more_gard** operation is expected to be a transfer of GARD ASA from the reserve address. However, no checks such as below was observed.

```
Gtxn[2].xfer_asset() == Int(stable_id)
```

However, based on the reserve_logic.py, there is no ASA other than GARD that the reserve can opt-into through an ASA transfer of 0 amount. As such, the reserve is not expected to hold any other ASA other than GARD and therefore the transaction is expected to fail. However, it is still recommended to implement explicit checks against the ASA ID and not just rely on implied behaviour.

Instance 2 – Liquidate

Affected File/Code

- price_validator.py
  - liquidate – Line 95-120

It was noted that during the **liquidate** operation, the smart contract lacked checks to ensure ASA being transferred has asset ID equal to **Int(stable_id)**.

The Gtxn[2], Gtxn[3] and Gtxn[4] of the transaction group for **liquidate** operation is expected to include the following.

- Gtxn[2]
  - From Liquidator
  - To the reserve

- Asset Transfer GARD
- Gtxn[3]
  - From Liquidator
  - To the treasury
  - Asset Transfer GARD
- Gtxn[4]
  - From Liquidator
  - To user address (CDP's original owner)
  - Asset Transfer GARD

The logic for validating the above transaction can be seen below but validation for ASA ID of Gtxn[2],[3] and [4] were not observed.

Lack of ASA ID checks during asset transfer may allow incorrect ASA to be used as a "payment" to the reserve, the treasury and CDP's original owner.

Instance 3 – GARD to Algo

Affected File/Code

- Treasury.py
  - GARD_TO_ALGO – Line 156-177

It was noted that during the GARD_TO_ALGO operation, the smart contract lacked checks to ensure ASA being transferred has asset ID equal to Int(stable_id).

The Gtxn[0] and Gtxn[1] of the transaction group **GARD_TO_ALGO** is expected to include the following.

- Gtxn[0]
  - From manager
  - Application Call to AppID Treasury
- Gtxn[1]
  - From manager
  - To Treasury
  - Asset Transfer of GARD

The logic for validating ASA ID of Gtxn[1] was not observed.

Lack of ASA ID checks during asset transfer may allow incorrect ASAs other than GARD to be used to exchange with Algo.

RECOMMENDATION

Review and enforce checks against the ASA ID of asset transfer transactions to ensure the correct ASA is being transferred.

REGRESSION TESTING COMMENTS

29th March 2022 – This issue is kept open.

Based on the GARD team feedback on implicit checks (ASA Opt-in) and additional validations added, the risk rating of this issue is updated to Low as it is still recommended to do explicit checks on ASA ID during asset transfer transactions. It is noted that based on the current logic, reserve logic would not have any ASA opt-in done other than GARD.

Instance 1 - **Mint More GARD**

Following feedback was provided by the GARD team.

As the reserve is only opted into GARD for ASA, it would not be possible for the reserve to hold or send any ASA other than GARD. It is also not possible to do an opt-in to ASA.

Instance 2 - Liquidate

Based on the commit c48ddf6811c7248192adb505fee7c8dae84ec37f, following additional checks have been added for Gtxn[3] and Gtxn[4].

price_validator.py – Line 111-112

```
Gtxn[3].xfer_asset() == Int(stable_id),
Gtxn[4].xfer_asset() == Int(stable_id),
```

Following feedback was provided by the GARD team.

ASA ID check for Gtxn[2] was not implemented as this is a transfer to the reserve, which would only be opted into the GARD ASA. As the reserve logic does not allow the reserve to opt-in to any other ASA other than GARD, any ASA transfer to reserve other than GARD would fail.

Instance 3 - **GARD to Algo**

Based on the commit c48ddf6811c7248192adb505fee7c8dae84ec37f, following additional check has been added for Gtxn[1].

Treasury.py – Line 161

```
Gtxn[1].xfer_asset() == stable_id,
```

VULNERABILITY REFERENCES

PyTeal Documentation – Algorand Standard Assets

https://developer.algorand.org/docs/get-details/asa/

1.10. Insufficient On-Chain Validation Against Altered Transactions

LOW RISK

VULNERABILITY TRACKING

STATUS: **Open**

BACKGROUND

Smart contracts often make use of front-end web applications to cater for wider pool of users through easy-to-use GUI-based interactions. However, as these front-end web applications are used to craft transaction groups based on user's input, if compromised, a maliciously altered version of transaction groups could be forwarded to or initiated by the actual user of Gard. Although the responsibility lies with the user to review the details of the transaction groups, it is recommended to have validations within the smart contract to protect the users against potentially damaging transaction groups, including the Algorand guidelines published.

DESCRIPTION

As the smart contract does not validate transaction fields such as sender, receiver, group_size, group_index, close_remainder_to, asset_close_to and rekeyTo of affected transactions, if the front-end web applications which exist to aid user interaction has been compromised, altered transactions which could be damaging to the users could be forwarded to the user for approval. If not reviewed thoroughly, such transaction groups damaging to the user could be approved. Although, the responsibility lies with the user who is approving the transaction group to review each transaction within the atomic group, it is still recommended to have on-chain validations as a safeguard. Do note that this issue only highlights scenarios where users could approve transaction groups which could be damaging to themselves.

Instance 1 - Closing CDP without Paying Devfee

Affected File

- cdp_escrow.py
  - RedeemStableNoFee - Line 95-103
- price_validtor.py
  - close_no_fee - Line 134-149

Affected transactions are noted below.

- Closing CDP without Paying Devfee
  - Gtxn[1]
    - rekey_to()
    - asset_close_to
  - Gtxn[3]
    - close_remainder_to()

- receiver()

For example, when the 'Closing CDP without Paying Devfee' transaction group is submitted, the user transfers the GARD debt to the GARD Reserve and is expected to receive the collateral back from the CDP contract account. However, due to lack of validation for the Gtxn[3].receiver() value, the collateral could be transferred to an arbitrary address other than the user address. Furthermore, the logic only checks if the Gtxn[3].close_remainder_to() is equal to a value other than Global.zero_address() which includes any arbitrary address that is valid. It is acceptable to omit checks on the "amount" as long as both receiver() and close_remainder_to() are set to userAddress.

Similar issues observed are highlighted as separate instances below.

Instance 2 - Open New Position Group A & B

Affected File

- price_validtor.py
- cdp_escrow.py

Current instance affects transaction groups noted below.

- Opening New Position A
    - Gtxn[0]
        - rekey_to()
        - close_remainder_to()
        - amount()
        - receiver()
        - group_size()
        - group_index()
    - Gtxn[1]
        - Global.group_size()
        - group_index()
    - Gtxn[2]
        - receiver()
        - rekey_to()
        - asset_close_to()
        - amount()
        - xfer_asset()
        - transaction_type
        - sender()
        - Global.group_size()
        - group_index()

- Opening New Position B
  - Gtxn[0]
    - Txn.group_index
  - Gtxn[1]
    - group_size()
    - group_index()
    - rekey_to()
    - close_remainder_to()
  - Gtxn[2]
    - Global.group_size()
    - group_index()
    - close_remainder_to()
    - rekey_to()
  - Gtxn[3]
    - receiver()
    - group_index()

Instance 3 - Voting from CDP & Key Registration

Affected File

- cdp_escrow.py
  - Vote - Line 27-51
- price_validtor.py
  - app_check - Line 200-209

Current instance affects transaction groups noted below.

- Voting from CDP
  - Gtxn[0]
    - receiver()
    - rekey_to()
    - close_remainder_to()
    - group_index()
  - Gtxn[1]
    - group_index()
- Voting from CDP with Arbitrary AppCall
  - Gtxn[0]

        - rekey_to()
        - close_remainder_to()
        - group_index()
    - Gtxn[1]
        - group_index()
    - Gtxn[2]
        - group_index()
- Key Registration
    - Gtxn[0]
        - rekey_to()
        - close_remainder_to()
        - group_index()
        - receiver()

Instance 4 - Change Pricing data

Affected File

- price_validator.py
    - change_price - Line 213-223

Current instance affects transaction groups noted below.

- Change Pricing Data
    - Txn
        - group_size()

Instance 5 - Add Voting App

Affected File

- Stake.py
    - add_vote_app - Line 82-92

Current instance affects transaction groups noted below.

- Add Voting App
    - Txn
        - rekey_to()
        - group_size()

Instance 6 - Remove Voting App

Affected File

- Stake.py

- remove_vote_app - Line 94-107

Current instance affects transaction groups noted below.

- Remove Voting App
    - Txn
        - rekey_to()
        - group_size()

Instance 7 - Lock Voting App

Affected File

- Stake.py
    - lock_vote_app - Line 109-117

Current instance affects transaction groups noted below.

- Lock Voting App
    - Txn
        - rekey_to()
        - group_size()

Instance 8 - Stake GAIN Token

Affected File

- Stake.py
    - stake - Line 123-138

Current instance affects transaction groups noted below.

- Stake
    - Gtxn[0]
        - rekey_to()
        - asset_close_to()
    - Gtxn[1]
        - group_index()

Instance 9 - Unstake GAIN Token

Affected File

- Stake.py
    - unstake - Line 141-159

Current instance affects transaction groups noted below.

- Unstake
    - Txn

- rekey_to()
- group_size()

Instance 9 - Swap Algo for GARD

Affected File

- treasury.py
  - ALGO_TO_GARD - Line 132-155

Current instance affects transaction groups noted below.

- Unstake
  - Gtxn[0]
    - rekey_to()
    - group_index()
  - Gtxn[1]
    - rekey_to()
    - group_index()
    - close_remainder_to()

Instance 10 - Swap GARD for Algo

Affected File

- treasury.py
  - GARD_TO_ALGO - Line 157-178

Current instance affects transaction groups noted below.

- Swap GARD for Algo
  - Gtxn[0]
    - rekey_to()
    - group_index()
  - Gtxn[1]
    - rekey_to()
    - group_index()
    - asset_close_to()

Instance 11 - Swap GAIN for Algo

Affected File

- treasury.py
  - claim - Line 108-130

Current instance affects transaction groups noted below.

- Swap GARD for Algo
    - Gtxn[0]
        - rekey_to()
        - group_index()
    - Gtxn[1]
        - rekey_to()
        - group_index()
        - asset_close_to()

Instance 12 - Vote

Affected File

- Vote_manager.py
    - send_vote- Line 86-107

Current instance affects transaction groups noted below.

- Vote
    - Txn
        - rekey_to()
        - group_size()

Instance 13 - Cancel

Affected File

- Vote_manager.py
    - cancel_vote- Line 110-117

Current instance affects transaction groups noted below.

- Cancel
    - Txn
        - rekey_to()
        - group_size()

Instance 14 - Init

Affected File

- Vote_manager.py
    - init_vote - Line 123-132

Current instance affects transaction groups noted below.

- Init
    - Txn

- rekey_to()
- group_size()

Instance 15 - Close

Affected File

- Vote_manager.py
  - close_vote - Line 135-142

Current instance affects transaction groups noted below.

- Close
  - Txn
    - rekey_to()
    - group_size()

Instance 16 - Minting More GARD from CDP Collateral

Affected File

- price_validator.py
- cdp_escrow.py
- treasury.py
- reserve_logic.py

Current instance affects transaction groups noted below.

- Minting More GARD from CDP Collateral
  - Gtxn[1]
    - rekey_to()
    - amount()
  - Gtxn[2]
    - receiver()

Instance 17 - Starting an Auction of the Collateral for GARD

Affected File

- price_validator.py
- cdp_escrow.py
- treasury.py

Current instance affects transaction groups noted below.

- Starting an Auction of the Collateral for GARD
  - Gtxn[0]
    - group_size()

Instance 18 - Liquidating a CDP

Affected File

- price_validator.py
- cdp_escrow.py
- treasury.py
- reserve_logic.py

Current instance affects transaction groups noted below.

- Liquidating a CDP
  - Gtxn[1]
    - receiver()
    - amount()
    - close_remainder_to()
      - should be set to Liquidator's address
  - Gtxn[2]
    - rekey_to()
    - asset_close_to()
    - group_size()
    - group_index()
  - Gtxn[3]
    - asset_amount()
      - The logic allows this value to be around 16.7% instead of the 20% of the remaining GARD after payment of GARD_DEBT to the reserve
    - rekey_to()
    - asset_close_to()
    - group_index()
    - group_size()
  - Gtxn[4]
    - asset_amount()
      - The logic allows this value to be around 83.3% instead of the 80% of the remaining GARD after payment of GARD_DEBT to the reserve. This allows liquidators who are liquidating their own CDP to pay less to the reserve and return more to their own account. This issue has been highlighted separately
    - rekey_to()
    - asset_close_to()

        - group_size()
        - group_index()

As Gtxn[1].receiver() and Gtxn[1].amount() do not have any validation and Gtxn[1].close_remainder_to() is only checked against a value other than Global.zero_address, it is possible for the insufficiently verified fields to be set to an address other than the address of the liquidator, who rightfully should claim the total algo balance of the CDP being liquidated.

Instance 19 - Closing a CDP and Paying a Devfee

Affected File

- price_validator.py
- cdp_escrow.py
- treasury.py
- reserve_logic.py

Current instance affects transaction groups noted below.

- Closing a CDP and Paying a Devfee
    - Gtxn[0]
        - group_index()
    - Gtxn[1]
        - rekey_to()
        - asset_close_to()
        - group_index()
        - group_size()
    - Gtxn[2]
        - group_index()
    - Gtxn[3]
        - group_index()
        - receiver()
        - closeRemainderTo
            - Validation only checks if it is not equal to the Global.zero_address and it can be set to any valid address, other than the owner of the CDP who is the rightful recipient of the Algos held by CDP as collateral

Instance 20 - Clearing Apps Opted into From CDP Besides the Validator to Prepare for Liquidation

Affected File

- price_validator.py
- cdp_escrow.py

Current instance affects transaction groups noted below.

- Clearing Apps Opted Into From CDP Besides the Validator to Prepare for Liquidation
  - Gtxn[0]
  - Gtxn[1]
  - Gtxn[2] - There is no logic shared for this transaction as this is an "Arbitrary transaction from Liquidator, only checks done are to make sure this isn't from the CDP being "cleared"
    - group_index()
    - group_size()
    - rekey_to()
    - type()

Instance 21 – Treasury – Pay Out

Affected File

- Treasury.py - Payout Line 68-104
  - Txn
    - rekey_to()

RECOMMENDATION

Appropriate transaction field validations can be added based on the context. Sample validations are noted below.

```
Txn.close_remainder_to() == user_address,
Txn.receiver() == user_address,
Txn.close_remainder_to()==Globla.zero_address()
Txn.rekey_to()==Global.zero_address()
Txn.asset_close_to()==Global.zero_address()
Txn.group_index()==Int(1)
Global.group_size()==Int(2)
```

REGRESSION TESTING COMMENT

Regression testing for this item has not yet been performed.

VULNERABILITY REFERENCES

Algorand Developer Portal – Guidelines

https://developer.algorand.org/docs/get-details/dapps/avm/teal/guidelines/

1.11. Lack of Checks for Locking Vote App

LOW RISK

VULNERABILITY TRACKING

STATUS: **Open**

BACKGROUND

Operations with condition requirements on local and global state of the smart contract should enforce checks before committing state changes to the contracts on blockchain to ensure unexpected outcome of such operations do not affect the smart contract's operations.

DESCRIPTION

Instance 1 – lock_vote_app

Affected File/Code

- Stake.py
  - lock_vote_app – Line 109-117

It was noted that during the **lock_vote_app** operation, the smart contract lacked checks to ensure it does not lock vote apps which has not even been added through **add_vote_app**. As observed in the lock_vote_app logic, only validation enforced is `Assert(isManager)`. It was also noted that the affected operation can only be done by the DAO Manager.

RECOMMENDATION

Review and enforce checks such as below before increasing the index of **Locked_votes.**

Example

```
App.globalGet(Bytes("Num_votes")) >  App.globalGet(Bytes("Locked_votes"))
```

REGRESSION TESTING COMMENTS

Regression testing for this item has not yet been performed.

VULNERABILITY REFERENCES

PyTeal Documentation – Assert

https://pyteal.readthedocs.io/en/stable/api.html?highlight=assert#pyteal.Assert.__init__

1.12. Redundant Code

OBSERVATIONAL

VULNERABILITY TRACKING

STATUS: **Open**

BACKGROUND

Use of redundant code in Algorand smart contracts may affect code readability, TEAL opcode computational cost and size limit. Algorand smart contracts are subjected to compilation size and opcode cost limitations such as below.

Smart Signatures

- 1000 bytes in size
- 20,000 in opcode cost

Smart Contracts

- 2KB Total for the compiled approval and clear program
- Size can be increased in 2KB increments, up to an 8KB for both approval and clear program
- 700 for single transactions, for group transactions, opcode cost is pooled

DESCRIPTION

Instance 1:

Affected File

- Vote_manager.py
    - valid_vote_check - Line 83-85

Based on the design decision taken, the following subroutine found in Vote_manager.py at 83-85 can be removed as it always returns True.

Vote_manager.py – Line 83-85

```
@Subroutine(TealType.uint64)
      def valid_vote_check(vote: TealType.bytes):
            # Any vote is valid, users should be careful when sending a vote
            return Int(1)
```

Instance 2:

Affected File

- treasury.py
    - valid_vote_check - Line 83-85

Following duplicate code was observed.

treasury.py – Line 22 & 29

```
    manager_account = global_must_get(Bytes("Manager"), Int(2))
    # This account will change, the percentage might change
```

```
<REDACTED>
    founder_percent = Int(2)
    # This account will change, the percentage might change
    manager_account = global_must_get(Bytes("Manager"), Int(2))
```

RECOMMENDATION

Review the purpose of redundant code and remove them if not necessary.

REGRESSION TESTING COMMENT

Regression testing for this item has not yet been performed.

VULNERABILITY REFERENCES

CWE:
https://cwe.mitre.org/data/definitions/1041.html

1.13. Price Oracle and DAO Manager

OBSERVATIONAL

VULNERABILITY TRACKING

STATUS: **Open**

BACKGROUND

Since GARD is an algorithmic stable coin which could be considered as overcollateralized loans with Loan-to-Value(LTV) at ~71% and liquidation at collateralization below 115% where a stable Algorand Standard Asset GARD can be loaned or minted, the accuracy and availability of price oracle is critical to its ecosystem for upholding its value and price peg. For robustness, accuracy, and impartiality, it is recommended to use decentralized and independent oracles.

DESCRIPTION

It was noted that GARD makes use of price oracles for various purposes, including liquidation and minting. Since GARD is pegged to USD based on the liquidation mechanism of an overcollateralized loan/minting, accuracy of the ALGO/USD price is crucial to all participants of GARD.

During the review, it was noted that Algoracle price feed for Algo/USD (App ID # : 53083112) was used, which is currently only available in testnet and not mainnet. In addition, DAO manager is able to change the **PRICING_APP_ID** of the priceValidator contract after launch.

We noted following risks.

1. Inaccuracy, bias and low availability of the price oracle used for Algo/USD pair may allow unintended liquidations or other consequences based incorrect Algo/USD price available on-chain.
2. DAO Manager changing the **PRICING_APP_ID** to a biased, inaccurate, or malicious price oracle with either malicious intent or by human error.
3. Although first DAO Manager is set to a specific address during deployment, for subsequently voted DAO managers, there is no on-chain method available to ensure it is a multisignature account and such privileged role carries a high risk if it is assigned to a single signature account.

RECOMMENDATION

1. Once decentralized price oracle is available, switch to the accurate and impartial price oracle with high availability
2. Ensure first DAO Manager uses multi-sig for risk management
3. For subsequently voted DAO Managers, sufficient information should be provided to those voting to highlight the bigger risk carried by DAO manager candidates with single signature account compared to DAO manager candidates with multisignature accounts.

REGRESSION TESTING COMMENT

23rd March 2022 – This issue is kept open

Based on discussion with Algogard team, following updates were noted.

1. For the GARD launched on testnet provided for testing, Algoracle price feed for Algo/USD (App ID # : 53083112) was used.
2. Currently, there is no Algo/USD price oracle available in Algorand mainnet.
3. Upon launch of Algogard, a smart contract will be created to provide the Algo/USD price on-chain.
4. Price feed provided in the interim price feed oracle would be from different trusted sources with safeguards against unexpected errors or unavailability.
5. Once a decentralized price oracle which meets standards in terms of accuracy, availability and impartiality becomes available, the DAO manager can update the **PRICING_APP_ID** state within priceValidator.py via application call `Txn.application_args[0] == Bytes("ChangePricing").`
6. Number of "change_price" that can be done would also be increased to 4 in the future.
7. Privileges given to the DAO Manager relies on the consensus and a belief that stakeholders of GAIN would vote for the DAO Manager who would be also incentivized for the success of GARD and it is an inherent design decision made with risks known.

VULNERABILITY REFERENCES

CWE:
https://cwe.mitre.org/data/definitions/1041.html

1.14. Incorrect Comments

OBSERVATIONAL

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Comments in the code should provide accurate references to the reader and be in sync with the most updated business logic and specifications to avoid confusion.

DESCRIPTION

Instance 1

Affected File

- price_validator.py
  - Line 19-20
    - Price decrease from 115% to 100% instead of 105%
  - Line 75-76
    - Instead of 125% (5/4), it should be 115% (23/20)
  - Line 164-165
    - Instead of 150% (3/2), it should be 140% (7/5)
  - Line 192-197
    - Instead of 150% (3/2), it should be 140% (7/5)

price_validator.py

```
Line 19
# Decreases price linearly from 115% to 105% over 6 minutes
<REDACTED>
Line 75
# 5/4 x GARD > collateral x (USD/mAlgo)
<REDACTED>
Line 164
# 3/2 x GARD <= collateral x (USD/mAlgo)
<REDACTED>
Line 192
# 3/2 x GARD <= collateral x (USD/mAlgo)
<REDACTED>
```

RECOMMENDATION

Update incorrect comments so that accurate information can be referenced to the reader of the code.

REGRESSION TESTING COMMENT

28th March 2022 – This issue is closed.

Instance 1

Based on the commit abee2af8163c2e4be888c7cefc59e23cc368cb2c, incorrect comments have been updated with accurate information.

Price_validator.py – Line 19-20, 83-84, 172, 200

```
# Gets current price of collateral in the auction
# Decreases price linearly from 115% to 100% over 6 minutes
<REDACTED Line 21-82>
# 23/20 x GARD > collateral x (USD/mAlgo)
<REDACTED Line 85-171>
# 7/5 x GARD <= collateral x (USD/mAlgo)
<REDACTED Line 173-199>
# 7/5 x GARD <= collateral x (USD/mAlgo)
```

VULNERABILITY REFERENCES

CWE:
https://cwe.mitre.org/data/definitions/1041.html

5. APPENDIX

DISCLAIMER

The material contained in this document is confidential and only for use by the company receiving this information from Vantage Point Security Pte. Ltd. (Vantage Point). The material will be held in the strictest confidence by the recipients and will not be used, in whole or in part, for any purpose other than the purpose for which it is provided without prior written consent by Vantage Point. The recipient assumes responsibility for further distribution of this document. In no event shall Vantage Point be liable to anyone for direct, special, incidental, collateral or consequential damages arising out of the use of this material, to the maximum extent permitted under law.

The security testing team made every effort to cover the systems in the test scope as effectively and completely as possible given the time budget available. There is however no guarantee that all existing vulnerabilities have been discovered due to the nature of manual code review. Furthermore, the security assessment applies to a snapshot of the current state at the examination time.

SCOPE OF AUDIT

Vantage Point reviewed the smart contracts underlying codebase to identify any security or economic flaws, or non-compliance to Algorand best practices. The scope of this review included the following test-cases and audit points.

- Insufficient Sender Address Validation for Privileged Operations
- Lack of Validation for Validity of Referenced States from External Applications
- Insufficient Validation of Transaction Fields and Types
- Validation of RekeyTo address for non-rekeying transactions
- Validation of CloseRemainderTo and AssetCloseTo for non-closing transactions
- Validation of Asset Identifier for Asset Transfer Transactions
- Validation of GroupIndex and GroupSize for Transaction Groups
- Incorrect Order of Operations
- Smart Contract Versions
- Incorrect Use of ScratchVar, Local and Global States
- Flawed/Inaccurate Logical/Mathematical Operations
- Overflow or Underflow Possibilities based on Valid Argument Ranges
- Validation of user-supplied Application Arguments
- Use of Multisignatures for Privileged Accounts
- Other known Algorand Best Practices and Guideline