SMART CONTRACT AUDIT PUBLIC REPORT

Smart Contract Audit for VerdeX Finance
VPQ-20220034
Verde Labs
9th March 2022

VANTAGEPOINT
Security at the Speed of Development

TABLE OF CONTENTS

1. Executive Summary

Overview

Vantage Point Security Pte Ltd was engaged by Verde Labs to conduct an Algorand smart contract security review of VerdeX Finance Smart Contract to identify security vulnerabilities, weaknesses and any instances of non-compliance to best practices within the smart contract. Testing commenced on the 21st February 2022 and was completed on the 8th March 2022.

Algorand smart contract security review was conducted based on the following materials provided.

- Documents
  - VerdeX Starter
    - https://docs.google.com/document/d/18FNjHOLeQ6k7wpUyhLVNSiCChgWi_5EgYTEGeWxVKIg/edit
- PyTeal Code
  - Private Repo
    - https://github.com/Verdex-Labs/launchpad-contract
      Commit ID de7dd291b4321bfc9cd182d211939e695ebf818b

Vantage Point performed this review by first understanding provided documents and code to understand the high-level business logic of the VerdeX Finance and different type of auctions. Based on this understanding and discussions with the Verde Labs team we sought clarifications on potential issues, discrepancies, and flaws within the smart contract's logic.

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

The following noteworthy issues were identified during this review.

1. **Insufficient On-Chain Validation Against Altered Transactions**
   Transactions which do not involve rekeying and closing should validate closeRemainderTo, assetCloseTo and rekeyTo against the Global Zero Address to ensure unexpected rekeying or closing is approved due to user's mistake.
2. **Missing Checks for .group_size() and .group_index()**
   A lack of validation against group transaction's size and transaction index may allow rogue transactions to be inserted into a legitimate group transaction or have unexpected transactions to be routed smart contract's logic.

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	**0**	**0**	**0**
Medium	**0**	**0**	**0**
Low	**2**	**0**	**2**
Observational	**3**	**0**	**3**
Summary	**5**	**0**	**5**

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	Alex Ng
Contact Email	an@verdexlabs.com
Application Name	VerdeX Finance
Testing Period	21st February 2022 – 8th March 2022
GIT Commit ID	de7dd291b4321bfc9cd182d211939e695ebf818b
Items Completed	Vantage Point completed the Smart Contract Security Review for below files • IFO.py • dutchAuction.py

Component	Review Type	Status
Algorand Smart Contract	Smart Contract Security Review	Completed
Algorand Smart Contract	Smart Contract Security Review Retest	Completed

VERSION HISTORY

Date	Version	Release Name
7th March 2022	v0.1	Draft
8th March 2022	v0.2	Draft
8th March 2022	v0.3	Draft
9th March 2022	v0.4	Draft
10th March 2022	v1.0	Final
11th March 2022	v1.1	Retest
25th March 2022	v1.2	Retest
30th March 2022	v1.3	Index Update

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. Smart Contract Audit for VerdeX Finance		LOW RISK	⚠
1.1. Insufficient On-Chain Validation Against Altered Transactions	Closed	LOW RISK	⚠
1.2. Missing Checks for .group_size() and .group_index()	Closed	LOW RISK	⚠
1.3. Unnecessary Checks or Validations	Closed	OBSERVATIONAL	ⓘ
1.4. Inaccurate Comments	Closed	OBSERVATIONAL	ⓘ
1.5. Difference in Design Specification and Actual Implementation	Closed	OBSERVATIONAL	ⓘ

4. Detailed Description of Vulnerabilities

1. Smart Contract Audit for VerdeX Finance LOW RISK

1.1. Insufficient On-Chain Validation Against Altered Transactions

LOW RISK

VULNERABILITY TRACKING

STATUS: **Closed**

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, unless it is explicitly allowed based on the design.

DESCRIPTION

As the smart contract does not validation transaction fields such as close_remainder_to, asset_close_to and rekeyTo fields for transactions from the user, if the front-end web applications which exist to aid user interaction has been compromised, altered transactions which could be damaging to the user could be forwarded to the user for approval. If not reviewed thoroughly, such transaction groups damaging to the user could be approved, unless this is an intended features or requirement. 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:

Affected File/Code

- dutchAuction.py
    - on_creation - Line 140-157

  - on_setup - Line 173-225
  - on_withdraw – Line 284-312
  - on_finalize – Line 316-345
  - on_cancel - Line 348-361
  - on_recover_wrong_tokens - Line 369-379
  - on_transfer_ownership - Line 384-393
  - on_transfer_proxy_admin - Line 398-402
  - on_set_auction_wallet - Line 407-416
  - on_final_admin_withdraw – Line 423-463
  - on_update – Line 481-484
  - OnComplete.OptIn – Line 488
  - on_commit - Line 231-279

- IFO.py
  - on_initialize - Line 173-189
  - on_setup_pool - Line 197-235
  - on_deposit_pool - Line 241-292
  - on_harvest_pool – Line 294-333
  - on_final_withdraw - Line 341-366
  - on_recover_wrong_tokens - Line 374-384
  - on_transfer_ownership - Line 386-395
  - on_transfer_proxy_admin - Line 397-401
  - OnComplete.OptIn – Line 429
  - on_update – Line 419-422

It was noted that non-closing transactions did not have validation of rekeyTo, CloseRemainderTo and AssetCloseTo against the Global.zero_address(). This only affects the user accounts while there is no impact to the smart contracts in scope.

Other instances of same issue have been highlighted under affected file/code.

RECOMMENDATION

Verify that non-rekeying transactions have RekeyTo property set to global zero address:

```
no_rekey_addr = Txn.rekey_to() == Global.zero_address()
```

Implement validations for CloseRemainderTo and AssetCloseTo fields for non-closing payment or asset transfer transactions.

```
no_close_remainder_to_addr = Txn.close_remainder_to() == Global.zero_address()
no_asset_close_to_addr = Txn.asset_close_to() == Global.zero_address()
    return And(
        no_close_remainder_to_addr,
        no_asset_close_to_addr
    )
```

REGRESSION TESTING COMMENT

24th March 2022 – This issue is closed.

The updated VerdeX Starter v1.1.0 documentation stated that the transactions fields CloseRemainderTo, AssetCloseTo and RekeyTo are intentionally left configurable as a feature for users, however comes with the responsibility on their part to exercise caution when making such transactions.

VULNERABILITY REFERENCES

Algorand Developer Docs – Transaction Reference - Payment Transaction: https://developer.algorand.org/docs/get-details/transactions/transactions/#closeremainderto

Algorand Developer Docs – Transaction Reference – Asset Transfer Transaction: https://developer.algorand.org/docs/get-details/transactions/transactions/#closeassetto

Algorand Developer Docs – Transaction Reference - RekeyTo https://developer.algorand.org/docs/get-details/transactions/transactions/#rekeyto

1.2. Missing Checks for .group_size() and .group_index()

LOW RISK

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

For Algorand smart contracts, multiple transactions can be grouped as a single group transaction and sent to the smart contract. If `Global.group_size()` is not verified, rogue transactions can be included as one of group transactions and sent to the smart contract and cause an unexpected behaviour.

DESCRIPTION

Instance 1 - Missing Global.group_size() Checks

Affected Files/Code

- dutchAuction.py
  - on_creation - Line 140-157
  - on_setup - Line 173-225
  - on_cancel - Line 348-361
  - on_recover_wrong_tokens - Line 369-379
  - on_transfer_ownership - Line 384-393
  - on_transfer_proxy_admin - Line 398-402
  - on_set_auction_wallet - Line 407-416
  - OnComplete.OptIn – Line 489
  - on_update – Line 481-484
- IFO.py
  - on_initialize - Line 173- 189
  - on_setup_pool - Line 197- 235
  - on_final_withdraw - Line 341- 366
  - on_recover_wrong_tokens - Line 374- 384
  - on_transfer_ownership - Line 386- 395
  - on_transfer_proxy_admin - Line 397- 401
  - onComplete.OptIn – Line 429
  - on_update – Line 419-422

It was noted that listed instances had missing checks for Global.group_size().

Instance 2 - Missing Checks for Txn.group_index()

Affected Files/Code

- dutchAuction.py
  - on_commit - Line 231- 279

- IFO.py
    - on_deposit_pool - Line 241-292

It was noted that above highlighted parts of the code did not have any checks for Txn.group_index() even though the transaction being validated is part of a group transaction.

RECOMMENDATION

Instance 1:

Implement checks for group_size of group transactions such as an example below.

Example - Instance 1

```
Global.group_size() == Int(3)
```

Instance 2:

Implement checks for group_index of individual transaction against the expected index.

Example - Instance 2

```
Txn.group_index() == Int(0)
```

REGRESSION TESTING COMMENT

24th March 2022 – Issue is closed.

Commit ID: 3ae6e188de4fb42a6166d10227a12d9954f81091

In the latest commit, it was observed that group_size() checks have been implemented for all affected transactions, except for the OnComplete.OptIn instances.

VerdeX team has commented that the team has intentionally omitted the `group_size()` check for the Application Calls with onComplete.OptIn as it is part of the system design that this operation can be called from a group transaction from any group size, which is stated in the updated documentation VerdeX Starter v1.1.0

Instance 1 - Missing Global.group_size() Checks

- dutchAuction.py
    - on_creation - Line 140-157 – Check added
    - on_setup - Line 173-225 – Check added
    - on_cancel - Line 348-361 – Check added
    - on_recover_wrong_tokens - Line 369-379 – Check added
    - on_transfer_ownership - Line 384-393 – Check added
    - on_transfer_proxy_admin - Line 398-402 – Check added
    - on_set_auction_wallet - Line 407-416 – Check added
    - OnComplete.OptIn – Line 489 – Can be included in group transaction of any size

  - on_update – Line 481-484 – Check added

- IFO.py
  - on_initialize - Line 173- 189 – Check added
  - on_setup_pool - Line 197- 235 – Check added
  - on_final_withdraw - Line 341- 366 – Check added
  - on_recover_wrong_tokens - Line 374- 384 – Check added
  - on_transfer_ownership - Line 386- 395 – Check added
  - on_transfer_proxy_admin - Line 397- 401 – Check added
  - onComplete.OptIn – Line 429 – Can be included in group transaction of any size
  - on_update – Line 419-422 – Check added

Instance 2 - Missing Checks for Txn.group_index()

- dutchAuction.py
  - on_commit - Line 231- 279 – Check added
- IFO.py
  - on_deposit_pool - Line 241-292 – Check added

VULNERABILITY REFERENCES

PyTeal Documentation – Global Group Size:
https://pyteal.readthedocs.io/en/stable/api.html#pyteal.Global.group_size

PyTeal Documentation – Global Group Index:
https://pyteal.readthedocs.io/en/latest/api.html#pyteal.TxnObject.group_in

1.3. Unnecessary Checks or Validations

OBSERVATIONAL ⓘ

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Use of redundant checks or unnecessary validations 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 Code

- dutchAuction.py - Line 145

It was noted that given the range of possible logical values for on_creation_auction_token_decimals.hasValue() and on_creation_auction_token_decimals.value(), following checks were found to be unnecessary.

As seen in dutchAuction.py Line 136-150, there are 2 possible scenarios.

Scenario 1 - ASA ID "on_creation_auction_token" Is Not A Valid ASA ID

- If supplied ASA ID "on_creation_auction_token" is not a valid ASA ID, AssetParam.decimals(on_creation_auction_token) returns a maybeValue with both .hasValue() and .value() equal to 0.

Scenario 2 - ASA ID "on_creation_auction_token" Is A Valid ASA ID

- If supplied ASA ID "on_creation_auction_token" is a valid ASA ID, AssetParam.decimals(on_creation_auction_token) returns a maybeValue with .hasValue() set to 1 and .value() set to the actual value.

As the purpose of validation here is to validate the ASA is a 6-decimal token, below check is unnecessary since on_creation_auction_token_decimals.value() will be equal to 0 and the check will fail regardless of the on_creation_auction_token_decimals.hasValue().

Instance 2:

Affected Code

- dutchAuction.py - Line 183

It was noted that given other existing checks in on_setup, below check was identified as unnecessary.

dutchAuction.py - Line 175 ~ 196

```
on_setup_start_time < Int(10_000_000_000),
on_setup_end_time < Int(10_000_000_000),
on_setup_end_time > on_setup_start_time,
```

As seen in the code snippet above, both on_setup_start_time and on_setup_end_time are compared against unix timestamp 10000000000 (Saturday, 20 November 2286 17:46:40) to cater for human errors or typos during on_setup.

Since below code ensures on_setup_start_time is less than on_setup_end_time, there is no need to check for *on_setup_start_time < Int(10_000_000_000).*

```
on_setup_end_time > on_setup_start_time,
```

RECOMMENDATION

Instance 1:

Below check can be removed from dutchAuction.py Line 145

```
on_creation_auction_token_decimals.hasValue(),
```

Instance 2:

Below check can be removed from dutchAuction.py Line 183

```
on_setup_end_time > on_setup_start_time,
```

Or alternatively, the following check can be removed from dutchAuction.py Line 182

```
on_setup_start_time < Int(10_000_000_000),
```

REGRESSION TESTING COMMENT

24th March 2022 – Issue is closed.

Commit ID: 3ae6e188de4fb42a6166d10227a12d9954f81091

Instance 1 - The redundant check has been removed from the commit noted above.

Instance 2 - The redundant check has been removed from the commit noted above.

VULNERABILITY REFERENCES

CWE – Use of Redundant Code:

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

1.4. Inaccurate Comments

OBSERVATIONAL

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

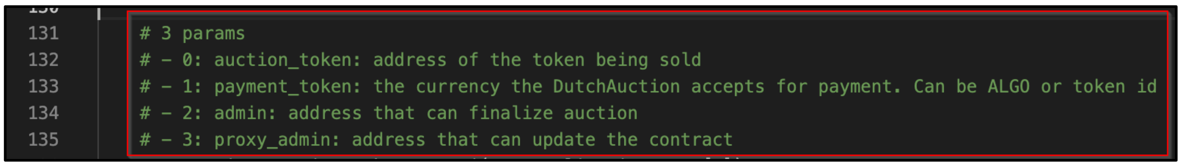
The source code contains comments that do not accurately describe or explain aspects of the portion of the code with which the comment is associated with.

DESCRIPTION

Affected File:

- dutchAuction.py – Line 131

Line 131 in dutchAuction.py states that there are 3 params in the call method `on_creation`. However, there are 4 params to be specified by the method:



```
131    # 3 params
132    # - 0: auction_token: address of the token being sold
133    # - 1: payment_token: the currency the DutchAuction accepts for payment. Can be ALGO or token id
134    # - 2: admin: address that can finalize auction
135    # - 3: proxy_admin: address that can update the contract
```

RECOMMENDATION

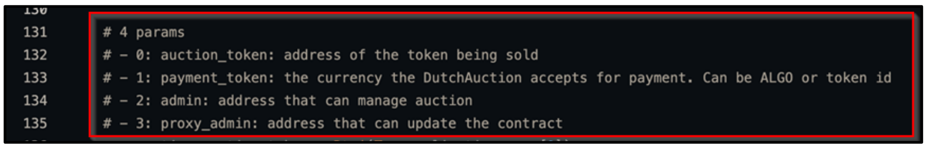
Review the affected lines of code and update the comments to accurately describe or explain the portion of the code that the comment is associated with.

REGRESSION TESTING COMMENT

24th March 2022 - Issue is closed.

Commit ID: 3ae6e188de4fb42a6166d10227a12d9954f81091

The comments at Line 131-135 have been updated to accurately reflect the actual implementation.



```
131    # 4 params
132    # - 0: auction_token: address of the token being sold
133    # - 1: payment_token: the currency the DutchAuction accepts for payment. Can be ALGO or token id
134    # - 2: admin: address that can manage auction
135    # - 3: proxy_admin: address that can update the contract
```

VULNERABILITY REFERENCES

CWE – Inaccurate Comments:

https://cwe.mitre.org/data/definitions/1116.html

1.5. Difference in Design Specification and Actual Implementation

OBSERVATIONAL ⓘ

VULNERABILITY TRACKING

STATUS: **Closed**

BACKGROUND

Often, functional design or specification documents are drafted prior to the actual development of smart contract in order to capture necessary requirements before so that the smart contract code can accurately reflect the logic built in a form of code. Once available to larger group of audiences or public members, there is a trust on whitepapers or documentations being in sync with the actual code running on Smart Contract and this forms a basis for any logical decision making for every participants. However, if there is a difference between what was documented and what the smart contract actually does, it may create a negative impact on the reliability of the protocol, losing confidence from the community and other stakeholders in blockchain.

DESCRIPTION

Instances noted below do not pose direct threat to the core availability and integrity of the smart contract but more of a difference in implementation specifics. Regardless, it is recommended to synchronize both document and smart contract to avoid any confusion.

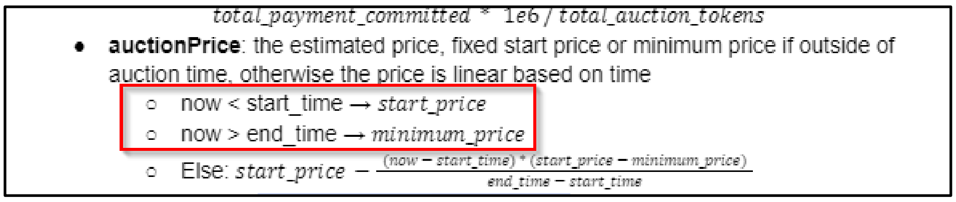
Instance 1

Affected File/Code

- dutchAuction.py – Line 71 ~ 74

According to *VerdeX Starter* page 10, the specification for start_price and minimum_price states that the latest timestamp must be < start_time for auctionPrice to be based on start_price, and latest timestamp < end_time for auctionPrice to be based on minimum_price.

Snippet from VerdeX Starter – Page 10



$total\_payment\_committed * 1e6 / total\_auction\_tokens$

- **auctionPrice**: the estimated price, fixed start price or minimum price if outside of auction time, otherwise the price is linear based on time
  - now < start_time → $start\_price$
  - now > end_time → $minimum\_price$
  - Else: $start\_price - \frac{(now - start\_time) * (start\_price - minimum\_price)}{end\_time - start\_time}$

However, in dutchAuction.py, the logic for *get_auction_price* was implemented differently from the documentation above as `<=` was used instead of `<`.

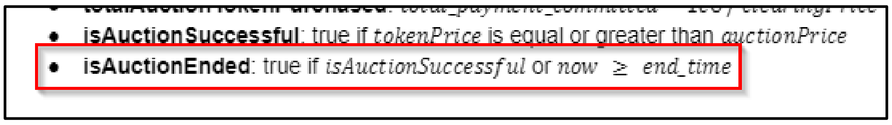
Instance 2

Affected File/Code

- dutchAuction.py – Line 306

According to *VerdeX Starter* page 10, the specification for *isAuctionEnded* states that the latest timestamp must be greater or equal to end time for the *isAuctionEnded* value to be true.

Snippet from VerdeX Starter – Page 10



- **isAuctionSuccessful**: true if $tokenPrice$ is equal or greater than $auctionPrice$
- **isAuctionEnded**: true if $isAuctionSuccessful$ or $now \geq end\_time$

However, in dutchAuction.py, the logic to determine if auction has ended was implemented differently from the documentation above as `>` was used instead of `>=`.

RECOMMENDATION

Synchronization between the actual implementation (smart contract code) and business logic (Functional Specifications/Documentation) is important as any discrepancy may result to a difference between the expected and the actual outcome. Decisions are made based on expected outcomes and if there is such a discrepancy, it risks confidence from community participants and other stakeholders who may heavily depend on the documented logic instead of the low-level code, which may not necessarily be available publicly.

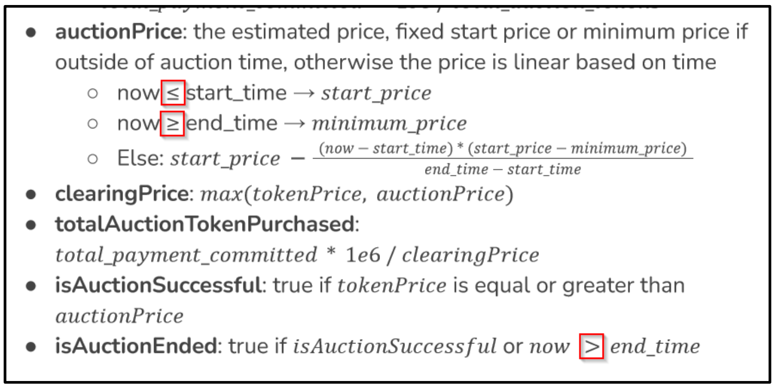
REGRESSION TESTING COMMENT

24th March 2022 – Issue is closed.

Commit ID: 3ae6e188de4fb42a6166d10227a12d9954f81091

The design specification for **auctionPrice** and **isAuctionEnded** has been updated in VerdeX Starter v1.1.0 documentation to match the implementation for:

- get_auction_price() in dutchAuction.py Line 72-73
- Assert check for if auction has finished in dutchAuction.py Line 306



- **auctionPrice**: the estimated price, fixed start price or minimum price if outside of auction time, otherwise the price is linear based on time
  - now $\boxed{\leq}$ start_time $\rightarrow start\_price$
  - now $\boxed{\geq}$ end_time $\rightarrow minimum\_price$
  - Else: $start\_price - \frac{(now - start\_time) * (start\_price - minimum\_price)}{end\_time - start\_time}$
- **clearingPrice**: $max(tokenPrice,\ auctionPrice)$
- **totalAuctionTokenPurchased**: $total\_payment\_committed\ *\ 1e6\ /\ clearingPrice$
- **isAuctionSuccessful**: true if $tokenPrice$ is equal or greater than $auctionPrice$
- **isAuctionEnded**: true if $isAuctionSuccessful$ or $now\ \boxed{>}\ end\_time$

VULNERABILITY REFERENCES

PyTeal Documentation – Lt:
https://pyteal.readthedocs.io/en/latest/api.html?highlight=Lt#pyteal.Lt

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 Guidelines