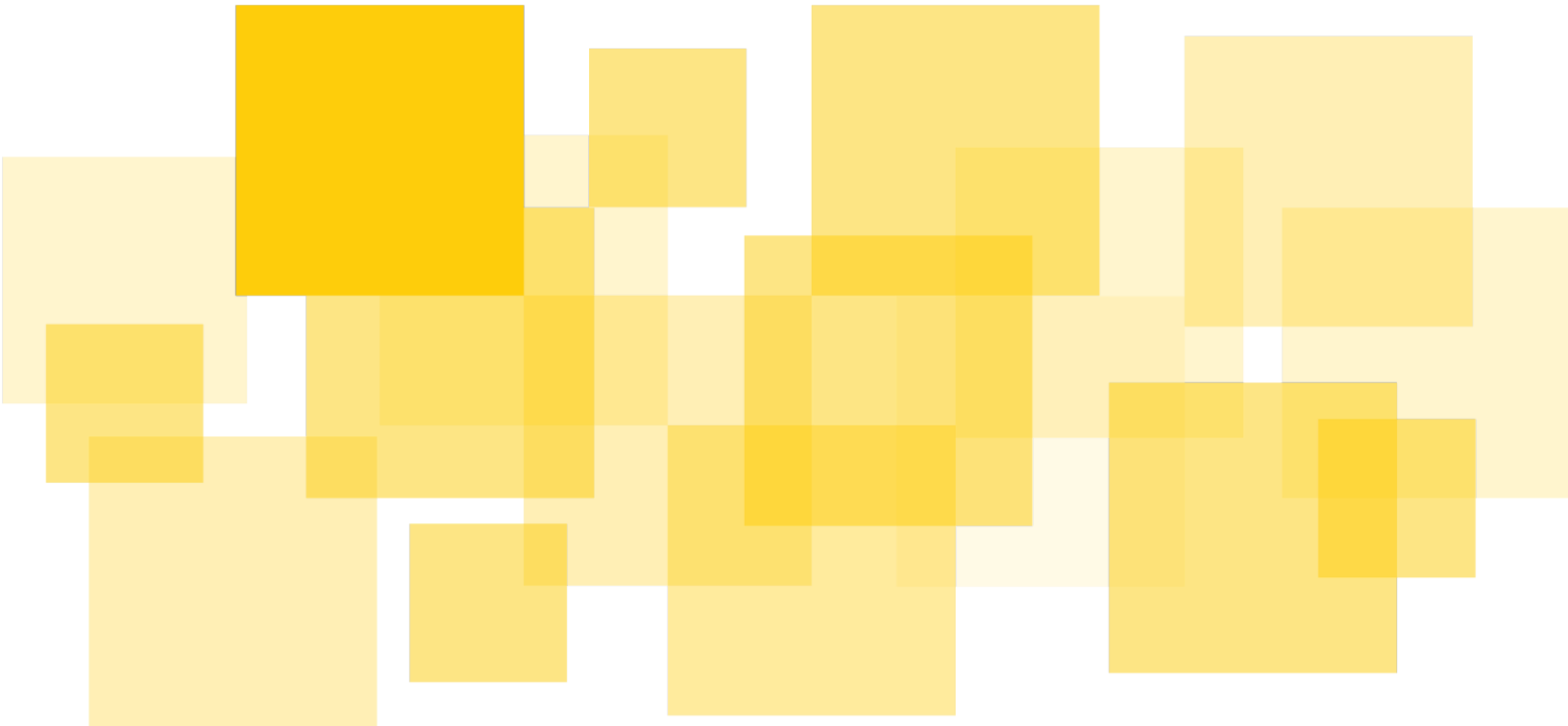


Security Audit Report (Partial)

Algodex Smart Contracts

Delivered: September 7, 2021

Minor amends: January 21, 2022



Prepared for Algodex by





Table of Contents

Summary

Goal

Scope

Methodology

Disclaimer

Findings

A01: Transaction fee is not checked in Buy order creation and cancellation

Description

Scenario

Recommendation

Status

A02: Application ID of Algo order book is not checked when cancelling a Buy order

Description

Scenario

Recommendation

Status

A03: Missing check for escrow owner in Algo order book

Description

Recommendation

Status

Informative Findings

B01: Centralized authority



Description

Recommendation

Status

B02: Redundant opcodes in Algo order book contract

Description

Recommendation

Status

B03: Late check of GroupSize in Algo escrow contract

Description

Recommendation

Status

B04: Redundant checks in Algo escrow contract

Description

Recommendation

Status

Appendix

Summary

Algodex engaged Runtime Verification Inc to conduct a review of the application's architecture and a security audit of their smart contracts. The objective was to review the contracts' business logic and implementations in TEAL and identify any issues that could potentially cause the system to malfunction or be exploited.

The audit was initially scheduled to take 6 calendar weeks, but had to be cut short at 2 weeks due to change of circumstances on Algodex's side. Therefore only a partial audit had been performed.

The audit was conducted by Georgy Lukyanov and Musab Alturki on the following artefacts provided by Algodex:

1. Algodex architecture (commit [48a502e3712f5900eb9266d42a524cb4f0456851](#)) — this private repository documents the architecture of Algodex in the form of textual description, diagrams and sample transactions;
2. Algodex SDK (commit [651f842a3aed7e752ab2ba1aa49b1fd908d04861](#)) — TEAL files specified in [Scope](#) were audited.

The audit led to identifying three potentially critical issues:

- A01: Transaction fee is not checked in Buy order creation and cancellation;
- A02: Application ID of Algo order book is not checked when cancelling a Buy order;
- A03: Missing check for escrow owner in Algo order book.

In addition, several informative findings and general recommendations.

Goal

The goal of the audit is twofold:

1. Review the high-level business logic (protocol design) of the Algodex system based on the provided documentation
2. Review the low-level implementation of the system given as smart contracts in TEAL

The audit focuses on trying to identify issues in the system's logic and its implementation that could potentially render the system vulnerable to attacks or cause it to malfunction. Furthermore, the audit highlights informative findings that could be used to improve performance and efficiency of the implementation and optimize its code size.

Scope

The scope of the audit is limited to the following artifacts:

1. The system's high-level documentation given by:
 - a. The [Overview document](#) which outlines the four smart contracts of the system and the possible execution scenarios specified by atomic transaction groups
 - b. Sequence [diagrams](#) documenting the atomic transaction groups
 - c. Sample [transactions](#)
2. A part of the system's smart contracts in TEAL. Specifically:
 - a. Stateful contract which handles the Algo order book (embedded in [dex_teal.js](#)) has been audited in full
 - b. Stateless contract which represents an Algo escrow account (embedded in [algo_delegate_template_teal.js](#)) has been audited partially until line 118 of 425.

The audit is limited in scope to the artifacts listed above. The other two smart contracts representing ASA order book and ASA escrow account were not audited due to early termination of the engagement. Off-chain and client-side portions of the codebase as well as deployment and upgrade scripts are not in the scope of this engagement.

Methodology

Algodev has provided us with a textual overview of the system and a number of sequence diagrams describing the interaction between the smart contracts and the users. Based on these materials, we rigorously reasoned about the business logic of the contracts, validating security-critical properties to ensure the absence of loopholes in the business logic.

After that, we thoroughly reviewed the contracts' source code to detect any unexpected (and possibly exploitable) behaviors. As TEAL is a relatively low-level language, we constructed different higher-level representations of the TEAL codebase, including:

1. automatically generated control-flow graphs using an extended version of the [Tealer Static Analyzer](#) tool,
2. manually writing a Haskell model of the contracts (based on the generated CFGs above), and then
3. cross-checking it with the high-level description of the system.

This approach enabled us to systematically check consistency between the logic and the low-level TEAL implementation.

We reviewed the TEAL guidelines published by Algorand to check for known issues.

Finally, given the nascent TEAL development and auditing community, we reviewed [this list](#) of known Ethereum security vulnerabilities and attack vectors and checked whether they apply to TEAL smart contracts; if they apply, we checked whether the code is vulnerable to them.



Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of this contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Findings

Findings presented in this section are issues that can cause the system to fail, malfunction, and/or be exploited, and should be properly addressed.

A01: Transaction fee is not checked in Buy order creation and cancellation

Description

The transaction fees are not checked to be within reasonable bounds when creating a Buy order (lines 48–149 of [algo_delegate_template_teal.js](#)) or cancelling a Buy order (lines 150–225 of [algo_delegate_template_teal.js](#)).

Scenario

Note that, currently, the logic indirectly guards against this vulnerability by always ensuring that the escrow is owned by the transaction sender. However, as the TEAL code for the contract evolves in future versions, this check could potentially not be enough anymore.

In case the ownership checks were not performed, an attacker could drain an escrow he does not own by creating a Buy order using that escrow and setting high transaction fees. The attacker does not get any benefit, but still can do some damage by sabotaging escrows.

Recommendation

Check that transaction fees are within reasonable bounds while placing or cancelling an order. Similar checks are performed while executing an order (lines 150–265):

```
txn Fee
int 1000
<=
```


Status

Addressed by Algodev.

A02: Application ID of Algo order book is not checked when cancelling a Buy order

Description

The Algo escrow contract intends to perform an application call to the Algo order book stateful contract, but does not check the `ApplicationID`.

Scenario

An attacker could craft a valid order cancellation transaction group with the application call transaction referring to an arbitrary application, thus bringing the Algo order book into an inconsistent state. The order book will have a record of a buy order which in fact no longer exists.

Recommendation

The Algo escrow must check the `ApplicationID` field of `TXN 0` refers to the Algo order book.

Status

Addressed by Algodev.

A03: Missing check for escrow owner in Algo order book

Description

The code fragment starting on line 272 of `dex_tea1.js` intends to validate that the Algo escrow contract and the Algo order book contract have consistent information on who is the escrow's owner. However, the logic does not perform the intended validation since the resulting boolean expression is popped from the stack instead of being asserted.

It is not clear whether this check is required, i.e. it looks like it may be a remnant of functionality that was not completely removed.

Recommendation

This check needs additional analysis by the Algodev team. If it is deemed necessary, then the `pop` instruction needs to be substituted with `assert`; otherwise the whole snippet can be removed.

Status

Addressed by Algodev.

Informative Findings

Findings presented in this section do not necessarily represent any flaw in the code itself. However, they indicate areas where the code may need some external support or areas where the code deviates from best practices. Due to discussion about previous issues hitting code size limits, we have included information on potential code size reductions.

B01: Centralized authority

Description

The governance of the Algodex smart contracts is fully in hands of a single centralised authority: the stateful contracts' creator.

Recommendation

For the mainnet release, the order book contracts should be governed by a multisig account, or, ideally, by some form of DAO.

Status

The developers were fully aware of the issue and are planning to eventually establish Algodex as a DAO.


B02: Redundant opcodes in Algo order book contract

Description

Use of instruction `app_local_get` is redundant when called as the next local state accessor after `app_local_get_ex`, since the latter already pushes the result of the former as the second return value onto stack. This redundancy could be observed around lines 208 and 274 of `dex_teal.js`.

Recommendation

For example, the first instance starting on line 200 of `dex_teal.js` could be rewritten as follows:

<pre>int 0 txn ApplicationID byte "creator" app_local_get_ex assert pop int 0 byte "creator" app_local_get txna Accounts 1 == assert</pre>		<pre>int 0 txn ApplicationID byte "creator" app_local_get_ex assert txna Accounts 1 == assert</pre>
--	--	---

Status

Addressed by Algodev.

B03: Late check of GroupSize in Algo escrow contract

Description

The contract starts with a loop that goes through the whole transaction group and checks the `RekeyTo` and `AssetCloseTo` properties. The checks that the transaction group is of valid size (up to four transactions) is performed later in the contract.

Recommendation

Introduce a blanket `assert` instruction that validates the group size before the loop to cut off large transaction groups early in the contract.

Status

Addressed by Algodev.

B04: Redundant checks in Algo escrow contract

Description

The Algo escrow stateless contract performs extensive validation of various transaction fields. However, some of the checks are not necessary and thus it is not clear why they are present: perhaps something else has been intended to be checked?

For example, in the file `algo_delegate_template_teal.js`, lines 66–68 check that the second transaction of the group is an application call, and then lines 74–76 check that the field `Amount` of this transaction contains 0. However, according to [Algorand Transaction Reference](#), an application call transaction does not contain the `Amount` field, and thus the result of accessing this field will always be 0.

Similar other checks are performed for fields that will not be present in transactions. Even though these checks are harmless, they are redundant, and thus are candidates for removal in case a reduction in the contract's code length is ever required.

Recommendation

Audit the redundant checks. Removing them is not strictly necessary.

Status

Taken into consideration by Algodev.

Appendix

We include here samples of artifacts created during the process of conducting this audit. One type of artifact is an automatically generated set of diagrams visualizing the control-flow graphs (CFG) of the programs' logic. The CFGs were generated using an extended version of the Tealer static analyzer by Trail of Bits available [here](#). Below (the trailing pages of the document) is a sample showing the generated CFG of the Algo order book contract.

Another type of artifact is a manually developed Haskell model that extracts the high-level logic implemented by the TEAL smart contracts. This model code makes it easier and more natural to reason about the business logic of the application and cross-check it against the intended behaviors of the different components of the system. For example, here is the part of the model that corresponds to the full order execution of an order (block 14 in the attached CFG):

```
{- ///////////////////////////////////////////////////////////////////
// EXECUTE (ORDER EXECUTION)
// WITH CLOSEOUT
/////////////////////////////////////////////////////////////////
// TXN 0 - ESCROW TO ORDERBOOK:
    transaction must be a call to a stateful contract
// TXN 1 - ESCROW TO SELLER:
    Payment transaction from this escrow to seller,
    with closeout to owner (buyer)
// TXN 2 - SELLER TO BUYER:
    Asset transfer from seller to owner of this escrow (buyer)
-}
scenarioExecuteFull ::
  ( MonadWriter (Set (KeyOp Value)) m
  , MonadFail m
  , MonadError StoreException m
  , MonadState (Store Value) m
  ) =>
  m Result
scenarioExecuteFull = do
  (VByte escrowAddress) <- read (KeyTxn TxnCurrent TxnSender)
  (VInt escrowBalance) <- read (KeyBalance escrowAddress)
  (VInt payAmount) <- read (KeyTxn (Txn 1) TxnAmount)
  assert 185 (escrowBalance - payAmount < 500000)
  (OnCompletion onCompletion) <- read (KeyTxn TxnCurrent TxnOnCompletion)
  (VInt groupSize) <- read (KeyGlobal GroupSize)
  txn0Type <- read (KeyTxn (Txn 0) TxnTypeEnum)
  txn1Type <- read (KeyTxn (Txn 1) TxnTypeEnum)
  txn2Type <- read (KeyTxn (Txn 2) TxnTypeEnum)
  assert 205 ( onCompletion == CloseOut
    && groupSize == 3
    && txn0Type == (VTxnType Appl)
    && txn1Type == (VTxnType Pay)
    && txn2Type == (VTxnType Axfer)
```

```

)
(VInt appID) <- read (KeyTxn TxnCurrent TxnApplicationID)
(VByte orderNumber) <- read (KeyTxn TxnCurrent (TxnApplicationArg 1))
-- fail if such order does not exist
assert 210 =<< app_local_get_ex escrowAddress appID orderNumber
-- fail if the local store for the escrow does not contain the "creator" key
assert 216 =<< app_local_get_ex escrowAddress appID "creator"
(VByte storedCreator) <- app_local_get escrowAddress "creator"
(VByte creator) <- read (KeyTxn TxnCurrent (TxnAccounts 1))
-- this assert was missing: check creator matches expectation.
assert 223 (storedCreator == creator)
assert 228 =<< app_local_get_ex escrowAddress appID "version"
-- close order: delete order from the order book
app_local_del escrowAddress creator
app_local_del escrowAddress "creator"
app_local_del escrowAddress "version"
pure (Approve "execute_with_closeout")

```

