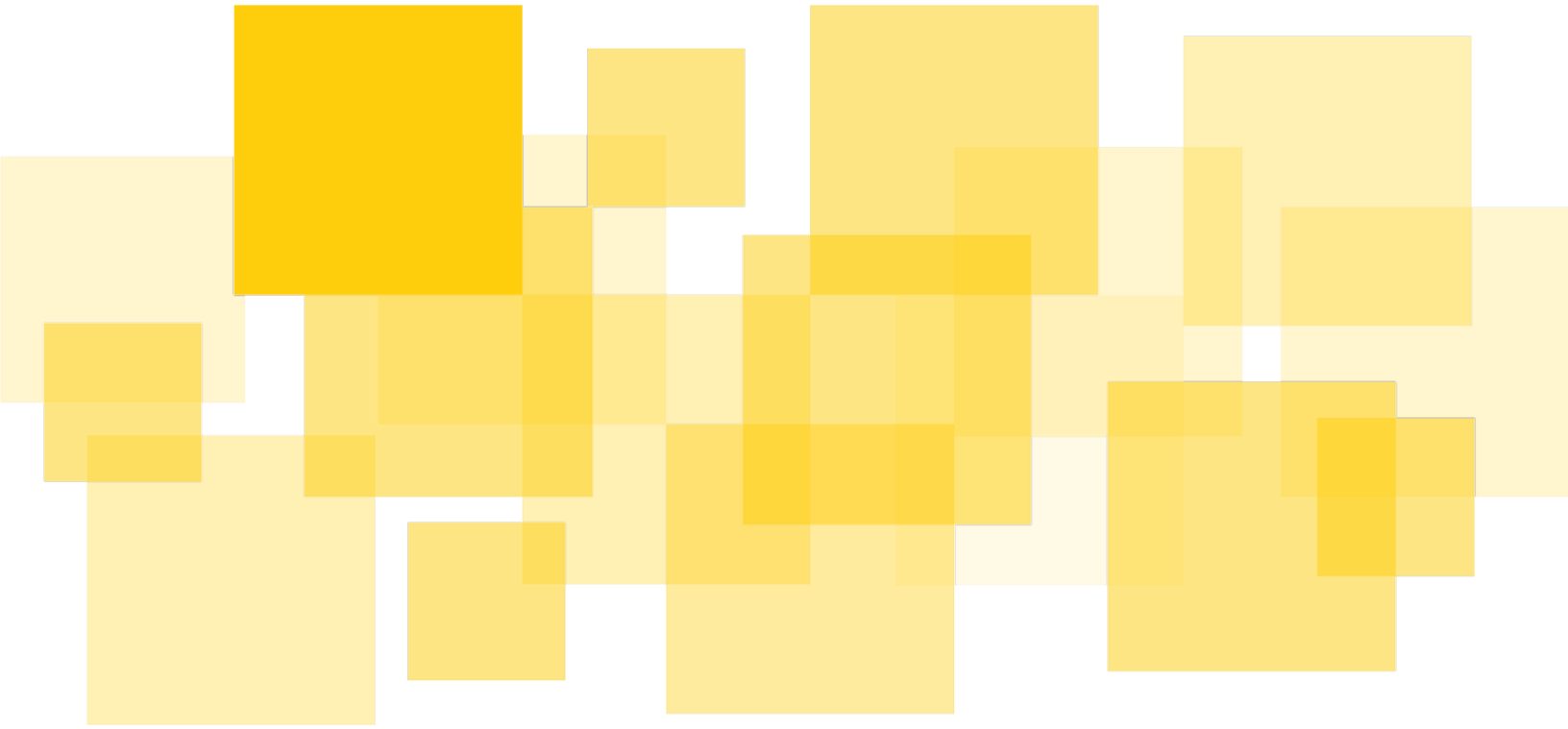
Security Audit Report

Algodex

Delivered: December 18, 2021



Prepared for Algodex by

runtime verification

Contents

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 implementation in Teal and identify any issues that could potentially cause the system to malfunction or be exploited.

The audit was conducted by Mihai Calancea over a period of 2 weeks (06 Dec 2021 - 17 Dec 2021).

The audit led to identifying one high severity issue and several informative findings and recommendations. Algodex have been receptive to our findings and suggestions and we have generally collaborated well with them.

Note that a previous audit was performed on the same scope by Georgy Lukyanov in September and produced an independent report[1].

Scope

The following `4` files were in scope for this audit

- The template for the *buy order escrow* `LogicSig`: algo_delegate_template_teal_v4.js.
- The template for the *sell order escrow* `LogicSig`: ASA_delegate_template_teal_v4.js.
- The order book smart contract for buy orders: dex_teal.js.
- The order smart contract book for sell orders: asa_dex_teal.js.

[1]The iteration of the scope analyzed here incorporates fixes for all issues flagged by the previous audit and refactorings for some of the recommendations.

Methodology

Algodex 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.

We then thoroughly reviewed the contracts' source code to detect any unexpected behaviors. As TEAL is a relatively low-level language, we used the Tealer Static Analyzer to generate control-flow graphs for an improved high-level view of the code.

The issue severity scale is mostly inspired by this classification. Issues that constitute more nuanced cases include additional justification. *Informative* issues concern general best-practice recommendations.

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 contracts only, and make no material claims or guarantees concerning the contracts' operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of these contracts.

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.

Issues and findings

A01. Missing on-chain safeguards in case of compromised front-end (High)

Context

We first describe a type of issue that can generally apply to Algorand applications.

Speaking from a general perspective, if user `Alice`[2] is the sender of at least one transaction in a given atomic group, then `Alice` can be perceived to be protected from harmful malformations of transaction fields belonging to this group. The reasoning is that Alice is rational and aware of the transaction group details[3], so would not sign off on a group which would harm her position.

Considering this, it can make sense for Algorand applications to not validate fields which may only harm `Alice`, if `Alice` must sign off on them. This can reduce the application's complexity and allows more freedom to `Alice`, which in some cases may be important.

However, this reasoning may be impractical when `Alice` interacts with the protocol via an off-chain front-end. It is usual for such front-ends to prepare transaction groups programatically for users to sign. In such cases `Alice` can become vulnerable if:

- The off-chain front-end is compromised at some point and maliciously alters transaction groups forwarded to users.
- `Alice` does not thoroughly check all groups forwarded to her before signing.

The impact of such a scenario can often be mitigated by performing additional on-chain sanity checks.

Issue

In the case of Algodex, two such sanity checks have been found to be missing:

- The *sell escrow* logic does not check that the receiver of the sold assets is the buyer (the sender of the payment transaction). If the front-end is compromised, it can serve transaction groups which redirect assets to the attacker.
- Certain operations on the protocol require that users refund the involved escrow account with the fees spent by the escrow during the operation. In the case of the *buy escrow* the amount refunded is checked to be at least `2000` algos, but no upper bound is checked. If the front-end is compromised, the attacker can set a very large `Amount` which may be to their advantage if they are in control of the refunded escrow.

Recommendation

- Check that the asset receiver is the sender of the payment transaction.

[2] Here we assume a human `Alice`. In particular, it's important that `Alice` is not a `LogicSig` escrow.
[3] This is especially convincing when `Alice` is plausibly the initiator of the transaction group.

- Check that the amount refunded is limited by a reasonable value.

Status

Solved.

A02. Inconsistency between transaction checks made in escrow and orderbook (Informative)

Context

Operations on the protocol are validated by both the logic of the involved escrow account and the logic of the involved order book. Sometimes they must make similar checks to correctly distinguish between different operations.

Finding

Certain checks are not specified identically in both contracts. Two examples:

- The order book distinguishes between operations by checking the application arguments of the incoming application call, but the escrow does not check these arguments (although it makes checks to other fields of the application call).
- The logic that checks the existence of the optional opt-in transaction in execution of sell orders is more restrictive in the escrow logic than in the order book.

This type of inconsistency opens up the possibility of malicious actors trying to confuse the escrow and order book regarding the transaction group structure, leading the order into an inconsistent, possibly insecure state.

We cannot currently describe such an attack, because the existing checks seem to offer sufficient redundancy. Nevertheless, we think the code can be made more robust and maintainable in this regard.

Recommendation

Increase symmetry between checks made on the escrow and the contract where applicable.

Status

The second example has been addressed by Algodex. More work in this direction will be done in a subsequent refactoring.

A03. Inconsistency in structure of transaction checks (Informative)

Context

The protocol needs to make different checks on transaction groups, depending on the intended operation and whether the checks are made by the escrow or the order book. While the logic

of these checks differs, it is good practice that they follow a consistent structure. For example, a possible structure is to check each transaction in order and check every relevant field in some fixed order of fields, which is consistent between different transactions and different contracts.

Having such a guideline makes the code more maintainable and facilitates detecting inconsistent or missing logic during development.

Finding

While the protocol code has some visible structure in this regard, it still presents several inconsistencies.

Recommendation

Choose a detailed guideline for performing checks and refactor contracts to reflect it. It can be useful to document the guideline in the comments of each individual file. Note that we do not recommend a particular guideline, just that it is consistently enforced in the codebase.

Status

Algodex intends to address this in a subsequent refactoring of the codebase.

A04. Opportunity to eliminate requirement of refund transactions (Informative)

Finding

This issue concerns the previously described practice of requiring users to refund the escrow account after a partial order execution. This was a good practice at the time the protocol was designed. However, the Algorand Virtual Machine now supports pooling of transaction fees. This means that the protocol can now:

- Eliminate the refund transaction from transaction groups.
- Check that fees spent by escrows are always `0`.

The user initiating the transaction is now required to adjust their fees to cover the total fee of the group adequately, otherwise the group will not pass.

Recommendation

Implement the above recommendation to simplify transaction group structure.

Status

Algodex intends to address this in a subsequent refactoring of the codebase.

A05. Opportunity for codebase improvement by using TEAL v5 (Informative)

Finding

The protocol can benefit from new features introduced by version `5` of the TEAL language. In particular, `InnerTransaction`s allow escrows to be implemented via `Applications`. This design can arguably simplify the protocol and increase its security. We note that implementing this change requires significant effort compared to previous recommendations.

Status

Algodex intends to address this in a subsequent refactoring of the codebase.