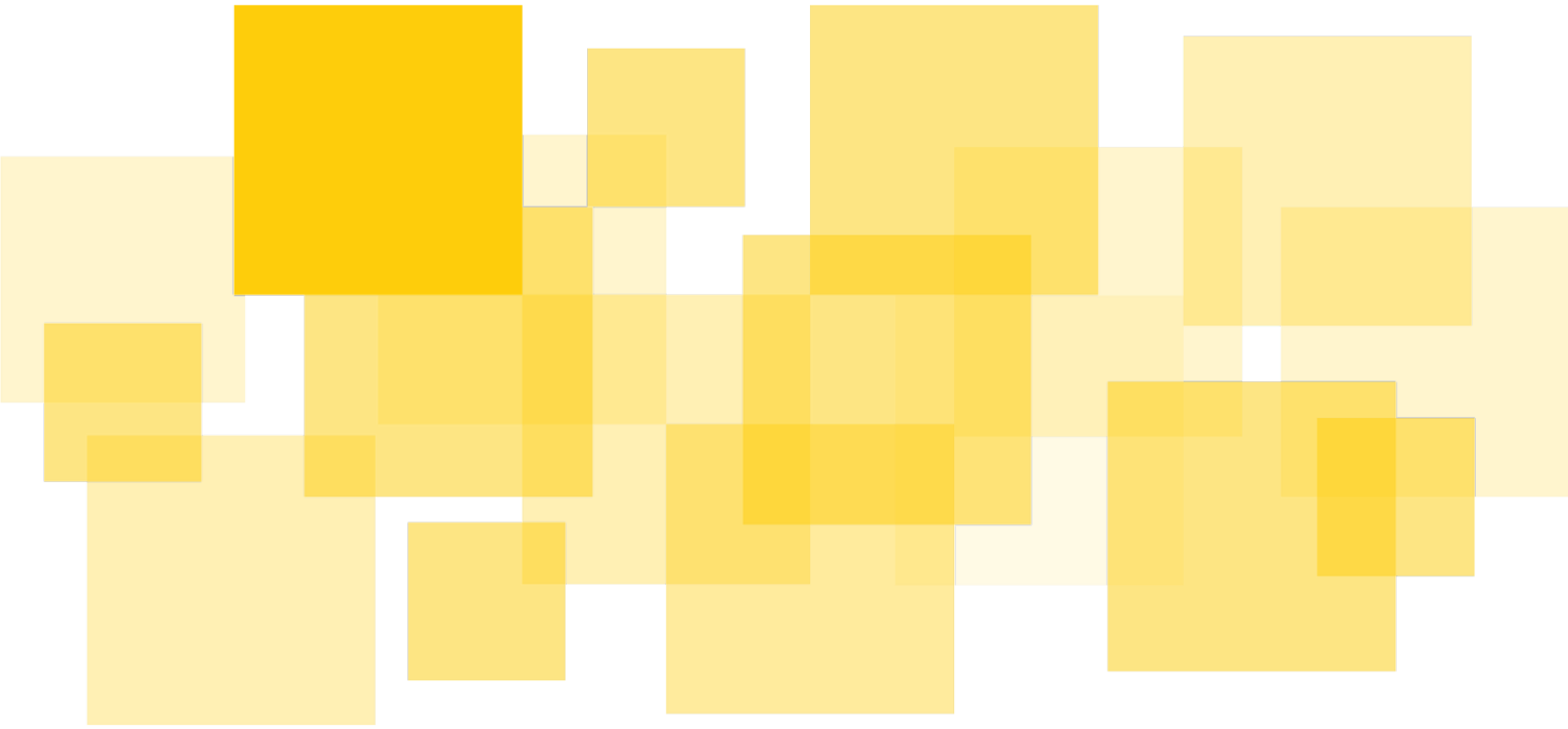
Security Audit Report

Algorand Governance Rewards

Delivered: September 30, 2021



Prepared for the Algorand Foundation by

runtime verification

Contents

Summary **2**
- Timeline . . . 2
- Scope . . . 2
- Findings . . . 2

Disclaimer **3**

Goals **4**

Scope **4**

Methodology **5**

Attack scenario analysis **6**
- A01. Obtaining control over the escrow . . . 6
- A02. Multiple claims by the same governor . . . 7
- A03. Malicious transaction draining escrow through transaction fees . . . 8

Additional Findings **8**
- B01. Delayed reward claim may be compromised . . . 8
- B02. Unreachable code in compiled `app_approval.teal` . . . 8

Appendices **9**

Summary

The Algorand Foundation engaged Runtime Verification Inc to conduct a security audit of the smart contracts implementing the Algorand Governance Rewards program.

The objective was to review the contracts' business logic and implementation in PyTeal and identify any issues that could potentially cause the system to malfunction or be exploited.

Timeline

The audit has been conducted under a tight time constraint over a period of 8 working days.

Scope

The audit was conducted by Georgy Lukyanov on the following artefacts provided by the Foundation:

- Rewards Application contracts/rewards_application.py maintains a list of governors and tracks their reward claims;
- Stateless Governance Escrow contracts/logicsig.py is an escrow account that holds the rewards and verifies payment transactions to the eligible governors.

Findings

Several potential attacks scenarios on the contracts were considered:

- A01. Obtaining control over the escrow
- A02. Multiple claims by the same governor
- A03. Malicious transaction draining escrow through transaction fees

All attack scenarios are either blocked by the validation logic, or the validation logic will be enhanced before deployment.

Additionally, we report several informative findings regarding the contracts design and implementation:

- B01. Delayed reward claim may be compromised
- B02. Unreachable code in compiled `app_approval.teal`

None of the informative findings constitute a threat to the rewards distribution process, but are still worth bringing to the attention of the Foundation and the wider Algorand community.

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.

Goals

The goals of the audit are:

- Review the architecture of the governance rewards smart contracts based on the provided documentation;
- Review the PyTeal implementation of the contracts and the compiled TEAL code to identify any programming errors;
- Cross check the compiled TEAL code of the contracts with the documented high-level design.

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.

Scope

We audit two smart contracts in the Algorand Governance Rewards repository as of commit 131c89304ea276c923ad025ec590ff4e1f551c3c:

- Rewards Application contracts/rewards_application.py maintains a list of governors and tracks their reward claims;
- Stateless Governance Escrow contracts/logicsig.py is an escrow account that holds the rewards and verifies payment transactions to the eligible governors.

Additionally, we use the design document for governance rewards smart contracts as a reference.

Methodology

The timeline was very tight, therefore we have only performed a best-effort audit.

Both smart contracts are implemented in PyTeal, a Python EDSL for writing TEAL programs. In order to exclude the PyTeal compiler form the trusted base, we compiled the contracts to TEAL and performed the audit on the compiled code.

We derived a control-flow graph for the contracts using a tool called Tealer. The graphs can be found in appendices to this document.

Basing on the CFG, we built a best-effort model of the contracts' semantics as a transition system embedded in the K Framework. The purpose of modelling was not to build a complete model, but rather to improve our understanding of the contracts' behaviour through the modelling process.

The model abstracts away the low-level details of the contracts, in particular the storage layout which is implemented as bit slices. We model storage as a traditional key-value data structure.

A combination of modelling and manual code review has enabled us to construct the attack scenarios presented above.

To encode the potential attack scenarios, we modified the provided test suite in `test_e2e.py` to include the malicious transaction groups. We used a local devnet setup provided by Algorand Foundation to execute the scenarios. We analyse the individual scenarios in further sections.

Attack scenario analysis

After reviewing the design and implementation of the contracts, we identified several possible attack scenarios, targeting two objectives:

- Stealing the funds from the rewards escrow;
- Disrupting the operation of the governance contracts;
- Partially burning the funds of the escrow via fees of the malicious transactions.

All these attacks have either proved to be impossible or the necessary mitigation measures were introduced.

However, the programming pattern we came across in the contracts' implementation has alerted us to perform additional checks, since it was divergent from the official Algorand developer guidelines. In particular, neither of the two contracts checks the size of an incoming transaction group. Therefore, we checked if a number of malicious transactions grouped with the valid ones could be approved. The rest of the section describes these malicious groups in more detail and confirms their denial by the contracts.

A01. Obtaining control over the escrow

The governance escrow will hold the rewards and issue payment to the governors. The Foundation plans to make this account non-participating to exclude it from the consensus and block it from earning rewards.

However, the validation code for this `KeyRegTxn` transaction does not check the `RekeyTo` field, enabling anyone to add an arbitrary authorised address for the escrow. This authorised address, if overlooked, may be used to steal the rewards after the escrow was funded by the rewards pool.

Furthermore, the logic does not check that the transaction group containing the `KeyReg` transaction is of size 1, i.e. an attacker could potentially slip in unchecked malicious transactions.

Recommendation

Check that the `RekeyTo` field of the transaction to make the escrow non-participating is set to `ZeroAddress`, thus blocking rekying completely. As per the design document, the escrow changes every governance period. Therefore there is no need to support rekeying it, since there is also no need to keep its public address static.

As for the group size check, since the escrow is only supposed to contain the minimal balance at that point, the attack is unlikely to succeed. Nevertheless, we suggest checking that `global GroupSize == 1` when handling the `KeyRegTxn` transaction to completely block any potential additional activity.

Status

The attack scenario has been brought to our attention by Shai Halevi — a member of the Algorand Foundation. This episode proves the vitality of close contact between the auditors and developers.

We assess the risk of success of the scenario as high, and the Foundation will ensure to fix the issue before deployment.

A02. Multiple claims by the same governor

A governor could attempt to submit a valid claim and an additional payment transaction, causing them to be payed twice.

We tried executing the attack on the sandboxed devnet provided by the Algorand Foundation. To our surprise, to make the transactions be even considered by the ledger, we had to make them unique, i.e. include random notes in the duplicate malicious transactions.

We tried submitting malicious groups of transactions of size 3 and 4:

Group 1. A valid group with an additional payment transaction: `[pay, appl, pay]`

Group 1 is rejected because there is no accompanying `ApplicationCallTxn` transaction for the extra pay transaction submitted. More specifically, the evaluation of the escrow's TEAL program failed on the additional payment transaction because an attempted out-of-bounds access by the `gtxns` opcode.

Group 2. A valid group with both transactions duplicated: `[pay, appl, pay, appl]`

Group 2 is rejected because by the TEAL approval program of the stateful smart contract. The effects of the first, valid, `ApplicationCallTxn` transaction are being applied to the tentative block; thus the governor's bit, tracking if the reward had already been payed, is already set to one, hence the rejection of the second `ApplicationCallTxn`.

Recommendation

We recommend introducing a `global GroupSize == 2` check into the escrow TEAL program. This would greatly simplify understanding of which transaction groups are considered valid, thus making it easier to ensure security.

Additionally, we suggest adding a negative integration test case to `test_e2e.py` describing the attack scenarios presented above.

A03. Malicious transaction draining escrow through transaction fees

Since the group size is not checked, another potential attack scenario was to submit a number of dummy `AssetCreation` transactions, thus forcing the escrow to pay network fees.

The scenario fails for the same reason that A02 does: the alignment of with an `ApplicationCallTxn` is not satisfied.

Recommendation

Same as for A02, we recommend adding a group size check to simplify the make the transaction validation logic easier to understand.

Additional Findings

B01. Delayed reward claim may be compromised

Description

The Algorand Governance FAQ, answering the question Q59, states that governors can claim their rewards at a later time since they may want to delay the claim for tax reasons. However, any third party account can trigger a claim for any eligible governor without their permission, causing the governor to get custody of their reward; thus potentially requiring them to report it to the tax authorities, depending on the jurisdiction.

Status

The Algorand Foundation is aware of this discrepancy. The incentive for a third party to trigger somebody else's claim is deemed to be negligible, since the said party does not gain anything. Allowing an external account being able to trigger an arbitrary governor's claim enables improving the user experience by providing a user-friendly web interface or a similar facility to claim rewards.

B02. Unreachable code in compiled `app_approval.teal`

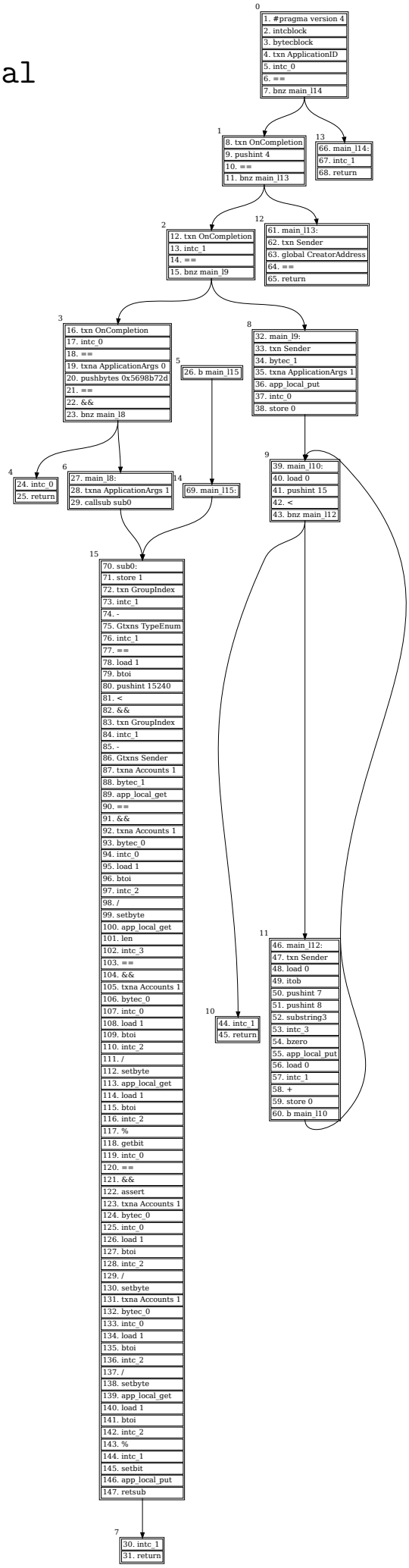
The compiled stateful contract contains an unreachable label as a second entry point to the "claim" subprogram. The label is an artefact of the PyTeal compiler and is harmless; hence this finding is purely informational.

Potentially, dead code could become a problem if the contract size approaches the limits prescribed by AVM.

Appendices

We include the control-flow graphs of the two contracts when compiled to TEAL. Please zoom the page in your PDF viewer to enlarge the graphs. You may want to omit the appendices when printing the report on paper.

app_approval.teal



logicsig.teal