
# Wxbooster

## AUDIT REPORT

Version 1.0.0

Serial No. 20220301000120013

Presented by Fairyproof

March 1st, 2022

# 01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Wxbooster project deployed on the Waves blockchain.

**Audit Start Time:**

Feb 23, 2022

**Audit End Time:**

March 1, 2022

**Audited Code's Github Repository:**

https://github.com/wxbooster/sc

**Audited Code's Github Commit Number When Audit Started:**

4aeb3dd0a3da33ee33013534c7bc2170f6c79e56

**Audited Code's Github Commit Number When Audit Ended:**

b93b239b3aabbbb946ab58910fccde686e3bc623

**Audited Source Files:**

The source files audited include all the files with the extension "ride" as follows:

```
sc/
├── giveaway.ride
├── lp.ride
└── proxy.ride

0 directories, 3 files
```

The goal of this audit is to review Wxbooster's implementation for its yield aggregator functions, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Wxbooster team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

# — Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from off-chain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# — The Wxbooster Team's Consent/Acknowledgement:

The audited materials of the project including but not limited to the documents, home site, source code, etc are all developed, deployed, managed, and maintained outside Mainland CHINA.

The members of the team, the foundation, and all the organizations that participate in the audited project are not Mainland Chinese residents.

The audited project doesn’t provide services or products for Mainland Chinese residents.

# — Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

## — Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

## — Documentation

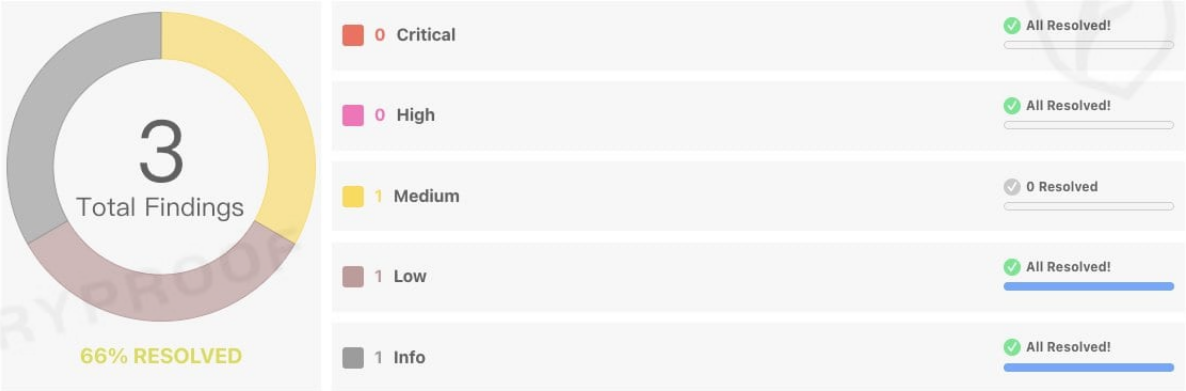
For this audit, we used the following sources of truth about how the yield aggregator system should work:

https://wxbooster.com

This was considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the yield aggregator team or reported an issue.

## — Comments from Auditor

| Serial Number | Auditor | Audit Time | Result |
| --- | --- | --- | --- |
| 2022030100012013 | Fairyproof Security Team | Feb 23, 2022 - March 1, 2022 | Medium |




Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, 1 risk of medium-severity, 1 risk of low-severity and 1 risk of informational-severity were discovered. Among these risks, 1 risk of low-severity and 1 risk of informational-severity were fixed, and 1 risk of medium-severity was confirmed.

# 02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

# 03. Introduction to Wxbooster

Wxbooster is a yield aggregator application deployed on the Waves blockchain.

# 04. Major functions of audited code

Users' crypto assets are eventually deposited into a third-party application deployed on `3PPNhHYkkEy13gRWDCaruQyhNbX2GrjYSyV` via this application and users will get rewards in both the token issued by Waves.Exchange and the token issued by this application.

The Wxbooster team takes away part of the reward issued by Waves.Exchange as a service fee and re-deposits it to a DApp deployed at `3PJL8Hn8LACaSBWLQ3UVhctA5cTQLBFwBAP` .

The `lp.ride` file contains user interfaces which users can use to stake or withdraw crypto assets or claim rewards. When a user stakes crypto assets, he/she can only claim his/her rewards after a staking period ends. If the user doesn't withdraw his/her staked assets after a staking period ends, the staked assets will be automatically rolled to the next staking period.

The `proxy.ride` file contains router interfaces. It has a whitelist. The whitelisted addresses can access `lp.ride` to stake or withdraw crypto assets or claim rewards. Note: the whitelist can be updated or disabled.

The `giveaway.ride` file contains a function that can be accessed by both the Wxbooster team and users.

In the existing implementation, every single staking crypto asset is handled by a single `lp.ride` contract. And a staking crypto asset cannot be handled by multiple `lp.ride` contracts otherwise distribution of rewards would be messed up.

The admin in this application is a 3-of-4 multi-sig wallet and it has the following rights:

- pausing staking,  pausing withdrawal or pausing claim of rewards
- updating the whitelist in the `proxy.ride` file
- upgrading smart contracts

# 05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Negative Number Check
- Use of BigInt
- Admin Rights
- Re-entrancy Attack
- Misuse of senderPublickey
- Validation of callerPublicKey
- Validation of Asset ID
- Function Check
- Upgradeability Check
- Functional Verification
- Design Vulnerability
- Use of strict

- Variable Setting

# 06. Severity level reference

Every issue in this report was assigned a severity level from the following:

**Critical** severity issues need to be fixed as soon as possible.

**High** severity issues will probably bring problems and should be fixed.

**Medium** severity issues could potentially bring problems and should eventually be fixed.

**Minor** severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

**Informational** is not an issue or risk but a suggestion for code improvement.

# 07. List of issues by severity

| Index | Title | Issue/Risk | Severity | Status |
| --- | --- | --- | --- | --- |
| FP-1 | Contract Upgradeable | Upgradeability Check | Medium | Confirmed |
| FP-2 | Missing Constrains for Variable | Variable Setting | Low | ✓Fixed |
| FP-3 | Missing Whitelist Verification | Design Vulnerability | Informational | ✓Fixed |

# 08. Issue descriptions

## [FP-1] Contract Upgradeable

Medium Confirmed

Issue/Risk: Upgradeability Check

Description:

With regard to Waves smart contracts, a Dapp script by default is upgradeable unless the script disables the feature. In the existing implementation, the Dapp script ran as admin and couldn't disable the upgradeability feature. Therefore all three `ride` contracts were upgradeable. Users who interacted with this application should acknowledge this.

Recommendation:

When the Wxbooster team intends to upgrade a smart contract, the contract should be audited prior to upgrading.

Update:

The Wxbooster team prefers to keep this feature in order for the application to better interact with third-party applications.

Status:

It has been confirmed by the Wxbooster team.

## [FP-2] Missing Constrains for Variable

Low ✓Fixed

Issue/Risk: Parameter Setting

Description:

In the `proxy.ride` file, the `setFee` function didn't have constraints for the settings of `lockFee` and `teamFee`. If they were set to 100%, users interests would be hurt.

Recommendation:

Consider adding constraints for this variable.

Status:

It has been fixed by the Wxbooster team. `10 <= lockFee <= 15` has been added for `lockFee` and `0 <= teamFee <= 5` has been added for `teamFee`.

## [FP-3] Missing Whitelist Verification

Informational ✓Fixed

Issue/Risk: Design Vulnerability

Description:

In the `proxy.ride` file, neither the `removeWhitelist` function nor the `setAllocation` function checked whether or not the handled address was a whitelisted address. Based on the implementation, only a whitelisted address would need these operations.

Recommendation:

Consider adding a directive to check whether or not the handled address is a whitelisted address.

Status:

It has been fixed by the Wxbooster team.

# 09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

Major Items:

- When throwing an error, it would be better to add a description, For example `throw("permission denied")`. This enhances the code's readability and makes it easier for debugging.
- Line 50 of the `proxy.ride` file, the code was `let balance = valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_balance"), 0) - amount`. It would be better to add `balance > 0` to prevent inappropriately withdrawing excessive LPs.

Misc:

- The whitelist shouldn't have duplicate items otherwise `assetId` would be messed up.
- Removing a whitelisted address should be very cautious. If it is inappropriately handled, a whitelisted user's assets may be lost.

# Appendix A: Source Code

`proxy.ride`:

```
{-# STDLIB_VERSION 5 #-}
{-# CONTENT_TYPE DAPP #-}
{-# SCRIPT_TYPE ACCOUNT #-}
```

```
let dAppAddress = Address(base58'3PPNhHYkkEy13gRWDCaruQyhNbX2GrjYSyV')
let wxAssetId = base58'Atqv59EYzjFGuitKVnMRk6H8FukjoV3ktPorbEys25on'
let wxbAssetId = base58'wxbassetid'
let giveawayAssetId = base58'Atqv59EYzjFGuitKVnMRk6H8FukjoV3ktPorbEys25on'
let wxLockAddress = Address(base58'3PJL8Hn8LACaSBWLQ3UVhctA5cTQLBFwBAP')
let teamFeeAddress = Address(base58'teamfeeaddress')

let adminPubKey1 = base58'pubkey1'
let adminPubKey2 = base58'pubkey2'
let adminPubKey3 = base58'pubkey3'
let adminPubKey4 = base58'pubkey4'

@Callable(i)
func stakeLP() = {
  let callerAddressString = toString(i.caller)
  let isWhitelisted =
valueOrElse(getBoolean(callerAddressString+"_whitelisted"), false)
  let lpAssetId = valueOrElse(getString(callerAddressString+"_assetId"), "")

  if(isWhitelisted && size(i.payments) == 1 && i.payments[0].amount > 0 &&
i.payments[0].assetId == fromBase58String(lpAssetId))
    then {

      strict stakeLPCall = invoke(dAppAddress, "stake", nil,
[AttachedPayment(i.payments[0].assetId, i.payments[0].amount)])
      let balance =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_balance"), 0) +
i.payments[0].amount

      (
        [
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_balance", balance)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func unstakeLP(amount: Int) = {
  let callerAddressString = toString(i.caller)
  let isWhitelisted =
valueOrElse(getBoolean(callerAddressString+"_whitelisted"), false)

  if(isWhitelisted)
    then {
      let lpAssetId = valueOrElse(getString(callerAddressString+"_assetId"), "")

      strict unstakeLPCall = invoke(dAppAddress, "unstake", [lpAssetId, amount],
nil)
      let balance =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_balance"), 0) -
amount

      (
        [
```

```
          ScriptTransfer(i.caller, amount, fromBase58String(lpAssetId)),
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_balance", balance)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func claim() = {
  let callerAddressString = toString(i.caller)
  let isWhitelisted =
valueOrElse(getBoolean(callerAddressString+"_whitelisted"), false)
  if(isWhitelisted)
    then {
      let lpAssetId = valueOrElse(getString(callerAddressString+"_assetId"), "")

      strict wxBalance = assetBalance(this, wxAssetId)
      strict claimWXCall = invoke(dAppAddress, "claimWx", [lpAssetId], nil)
      strict claimedWXAmount = assetBalance(this, wxAssetId) - wxBalance

      let totalClaimedWXPool =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_total_claimed_wx"),
0) + claimedWXAmount
      let totalClaimedWX = valueOrElse(getInteger(this, "total_claimed_wx"), 0)
+ claimedWXAmount

      let lockFeeRate = getIntegerValue(this, "lock_fee")
      let teamFeeRate = getIntegerValue(this, "team_fee")
      let lockFee = fraction(claimedWXAmount, lockFeeRate, 100)
      let teamFee = fraction(claimedWXAmount, teamFeeRate, 100)

      let totalCollectedFeePool =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_total_collected_fee")
, 0) + lockFee + teamFee
      let totalCollectedFee = valueOrElse(getInteger("total_collected_fee"), 0)
+ lockFee + teamFee
      let totalLockFeePool =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_total_lock_fee"), 0)
+ lockFee
      let totalLockFee = valueOrElse(getInteger("total_lock_fee"), 0) + lockFee
      let totalTeamFeePool =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_total_team_fee"), 0)
+ teamFee
      let totalTeamFee = valueOrElse(getInteger("total_team_fee"), 0) + teamFee

      strict wxLockCall = invoke(wxLockAddress, "increaseLock", [0],
[AttachedPayment(wxAssetId, lockFee)])
      let totalLockedWX = valueOrElse(getInteger("total_locked_wx"), 0) +
lockFee

      let wxbAllocation =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_wxb_allocation"), 0)
      let totalClaimedWXBPool =
valueOrElse(getInteger(callerAddressString+"_"+lpAssetId+"_total_claimed_wxb"),
0) + wxbAllocation
```

```
      let totalClaimedWXB = valueOrElse(getInteger("total_claimed_wxb"), 0) +
wxbAllocation

      (
        [
          ScriptTransfer(i.caller, (claimedWXAmount - lockFee - teamFee),
wxAssetId),
          ScriptTransfer(teamFeeAddress, teamFee, wxAssetId),
          ScriptTransfer(i.caller, wxbAllocation, wxbAssetId),
          IntegerEntry("total_locked_wx", totalLockedWX),
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_total_claimed_wx",
totalClaimedWXPool),
          IntegerEntry("total_claimed_wx", totalClaimedWX),
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_total_claimed_wxb",
totalClaimedWXBPool),
          IntegerEntry("total_claimed_wxb", totalClaimedWXB),
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_total_collected_fee",
totalCollectedFeePool),
          IntegerEntry("total_collected_fee", totalCollectedFee),
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_total_lock_fee",
totalLockFeePool),
          IntegerEntry("total_lock_fee", totalLockFee),
          IntegerEntry(callerAddressString+"_"+lpAssetId+"_total_team_fee",
totalTeamFeePool),
          IntegerEntry("total_team_fee", totalTeamFee)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func init() = {
  let initialized = valueOrElse(getBoolean("initialized"), false)
  if(!initialized)
    then {

      strict firstLock = invoke(wxLockAddress, "lock", [2102400],
[AttachedPayment(i.payments[0].assetId, i.payments[0].amount)])

      (
        [
          BooleanEntry("initialized", true),
          IntegerEntry("it_all_started_in_this_block", height),
          IntegerEntry("total_locked_wx", i.payments[0].amount),
          IntegerEntry("lock_duration", 2102400 + height),
          IntegerEntry("lock_fee", 15),
          IntegerEntry("team_fee", 0)
        ],
        unit
      )

    } else {
      throw("")
    }
}
```

```
@Callable(i)
func setFee(lockFee: Int, teamFee: Int) = {
  if(i.caller == this && lockFee <= 15 && lockFee >= 10 && teamFee <= 5 &&
teamFee >= 0)
    then {
      (
        [
          IntegerEntry("lock_fee", lockFee),
          IntegerEntry("team_fee", teamFee)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func whitelist(whitelistAddress: String, assetId: String, allocation: Int) = {
  if(i.caller == this)
    then {
      (
        [
          BooleanEntry(whitelistAddress+"_whitelisted", true),
          StringEntry(whitelistAddress+"_assetId", assetId),
          IntegerEntry(whitelistAddress+"_"+assetId+"_wxb_allocation",
allocation)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func removeWhitelist(whitelistAddress: String) = {
  let isWhitelisted = valueOrElse(getBoolean(whitelistAddress+"_whitelisted"),
false)
  if(i.caller == this && isWhitelisted)
    then {
      (
        [
          BooleanEntry(whitelistAddress+"_whitelisted", false)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func setAllocation(amount: Int, whitelistAddress: String, assetId: String) = {
  let isWhitelisted = valueOrElse(getBoolean(whitelistAddress+"_whitelisted"),
false)
  if(i.caller == this && isWhitelisted)
```

```
    then {
      (
        [
          IntegerEntry(whitelistAddress+"_"+assetId+"_wxb_allocation", amount)
        ]
      )
    } else {
      throw("")
    }
}

@Callable(i)
func giveaway() = {
  let callerAddressString = toString(i.caller)
  let isWhitelisted =
valueOrElse(getBoolean(callerAddressString+"_whitelisted"), false)

  if(isWhitelisted && size(i.payments) == 1 && i.payments[0].assetId ==
giveawayAssetId && i.payments[0].amount > 0 )
    then {

      strict wxLockCall = invoke(wxLockAddress, "increaseLock", [0],
[AttachedPayment(i.payments[0].assetId, i.payments[0].amount)])

      let totalLockedWX = valueOrElse(getInteger("total_locked_wx"), 0) +
i.payments[0].amount
      let totalGiveawayWX = valueOrElse(getInteger("total_giveaway_wx"), 0) +
i.payments[0].amount
      (
        [
          IntegerEntry("total_locked_wx", totalLockedWX),
          IntegerEntry("total_giveaway_wx", totalGiveawayWX)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func increaseLockDuration(duration: Int) = {
  if(containsElement([adminPubKey1, adminPubKey2, adminPubKey3, adminPubKey4],
i.callerPublicKey))
    then {

      strict increaseLockDurationCall = invoke(wxLockAddress, "increaseLock",
[duration], nil)

      let lockDuration = getIntegerValue("lock_duration") + duration

      (
        [
          IntegerEntry("lock_duration", lockDuration)
        ],
        unit
      )
    } else {
```

```
      throw("")
    }
}


@Verifier(tx)
func verify () = {
  let adminPubKey1Signed = if (sigVerify(tx.bodyBytes, tx.proofs[0],
adminPubKey1))
    then 1
    else 0
  let adminPubKey2Signed = if (sigVerify(tx.bodyBytes, tx.proofs[1],
adminPubKey2))
    then 1
    else 0
  let adminPubKey3Signed = if (sigVerify(tx.bodyBytes, tx.proofs[2],
adminPubKey3))
    then 1
    else 0
  let adminPubKey4Signed = if (sigVerify(tx.bodyBytes, tx.proofs[3],
adminPubKey4))
    then 1
    else 0
  ((adminPubKey1Signed + adminPubKey2Signed + adminPubKey3Signed +
adminPubKey4Signed) >= 3)
}
```

`lp.ride`

```
{-# STDLIB_VERSION 5 #-}
{-# CONTENT_TYPE DAPP #-}
{-# SCRIPT_TYPE ACCOUNT #-}

let lpAssetId = base58'assetid'
let wxAssetId = base58'Atqv59EYzjFGuitKVnMRk6H8FukjoV3ktPorbEys25on'
let wxbAssetId = base58'wxbassetid'
let proxyAddress = Address(base58'proxyaddress')
let roundHeight = 10080
let targetBlock = 3397380
let limit = 100000000000000000

let adminPubKey1 = base58'pubkey1'
let adminPubKey2 = base58'pubkey2'
let adminPubKey3 = base58'pubkey3'
let adminPubKey4 = base58'pubkey4'

@Callable(i)
func deposit() = {
  let isDepositLocked = valueOrElse(getBoolean("deposit_locked"), false)
  let previousBalance = valueOrElse(getInteger("total_balance"), 0)
  let isValidBlock = height < targetBlock
  let isInitialized = valueOrElse(getBoolean("initialized"), false)
  let isFirstBlockReached = height >= valueOrElse(getInteger("first_block"), 0)
```

```
  if(!isDepositLocked && isInitialized && isFirstBlockReached &&
size(i.payments) == 1 && i.payments[0].amount > 0 && i.payments[0].assetId ==
lpAssetId && isValidBlock && (previousBalance + i.payments[0].amount) <= limit)
  then {
    let startBlock = getIntegerValue("start_block")

    func getWalletPreviousShare() = {
      let lastDepositBlock =
valueOrElse(getInteger(toString(i.caller)+"_last_deposit_block"), 0)
      let lastWithdrawBlock =
valueOrElse(getInteger(toString(i.caller)+"_last_withdraw_block"), 0)

      let previousShare = if(lastDepositBlock != 0 && lastDepositBlock <
startBlock && lastWithdrawBlock < startBlock)
        then {
          let walletBalance =
valueOrElse(getInteger(toString(i.caller)+"_balance"), 0)
          toBigInt(walletBalance) * toBigInt(roundHeight)
        } else {

 parseBigIntValue(valueOrElse(getString(toString(i.caller)+"_"+toString(startBlo
ck)+"_share"), "0"))
        }

      previousShare
    }

    func getTotalPreviousShare() = {
      let lastDepositBlock = valueOrElse(getInteger("last_deposit_block"), 0)
      let lastWithdrawBlock = valueOrElse(getInteger("last_withdraw_block"), 0)

      let previousShare = if(lastDepositBlock != 0 && lastDepositBlock <
startBlock && lastWithdrawBlock < startBlock)
        then {
          let totalBalance = valueOrElse(getInteger("total_balance"), 0)
          toBigInt(totalBalance) * toBigInt(roundHeight)
        } else {

 parseBigIntValue(valueOrElse(getString(toString(startBlock)+"_total_share"),
"0"))
        }

      previousShare
    }

    let walletPreviousShare = getWalletPreviousShare()
    let depositShare = if(height > (startBlock + roundHeight))
      then {
        toBigInt(0)
      } else {
        toBigInt(i.payments[0].amount) * toBigInt((startBlock + roundHeight) -
height)
      }
    let walletShare = walletPreviousShare + depositShare

    let walletPreviousBalance =
valueOrElse(getInteger(toString(i.caller)+"_balance"), 0)
    let walletBalance = walletPreviousBalance + i.payments[0].amount
```

```
    let totalPreviousShare = getTotalPreviousShare()
    let totalShare = totalPreviousShare + depositShare

    let previousTotalBalance = valueOrElse(getInteger("total_balance"), 0)
    let totalBalance = previousTotalBalance + i.payments[0].amount

    strict depositCall = invoke(proxyAddress, "stakeLP", nil,
[AttachedPayment(i.payments[0].assetId, i.payments[0].amount)])

    let previousWalletActivity = match
(getString(toString(i.caller)+"_activity")) {
      case p: String => true
      case n: Unit => false
    }
    let walletActivity = if(previousWalletActivity)
      then {
        if(!contains(getStringValue(toString(i.caller)+"_activity"),
toString(startBlock)))
          then {
            getStringValue(toString(i.caller)+"_activity") + "_" +
toString(startBlock)
          } else {
            getStringValue(toString(i.caller)+"_activity")
          }
      } else {
        toString(startBlock)
      }

    let firstDeposit = match (getInteger(toString(i.caller)+"_first_deposit")) {
      case p: Int => p
      case n: Unit => startBlock
    }

    (
      [
        IntegerEntry(toString(i.caller)+"_balance", walletBalance),
        StringEntry(toString(i.caller)+"_"+toString(startBlock)+"_share",
toString(walletShare)),
        IntegerEntry(toString(i.caller)+"_"+toString(startBlock)+"_balance",
walletBalance),
        StringEntry(toString(i.caller)+"_activity", walletActivity),
        IntegerEntry(toString(i.caller)+"_last_deposit_block", startBlock),
        IntegerEntry(toString(i.caller)+"_first_deposit", firstDeposit),
        IntegerEntry("total_balance", totalBalance),
        StringEntry(toString(startBlock)+"_total_share", toString(totalShare)),
        IntegerEntry("last_deposit_block", startBlock)
      ],
      unit
    )
  } else {
    throw("")
  }
}

@Callable(i)
func withdraw(amount: Int) = {
  let isWithdrawLocked = valueOrElse(getBoolean("withdraw_locked"), false)
```

```
  let walletPreviousBalance =
valueOrElse(getInteger(toString(i.caller)+"_balance"), 0)
  if(!isWithdrawLocked && amount > 0 && walletPreviousBalance > 0 && amount <=
walletPreviousBalance)
    then {
      let startBlock = getIntegerValue("start_block")

      func getWalletPreviousShare() = {
        let lastDepositBlock =
valueOrElse(getInteger(toString(i.caller)+"_last_deposit_block"), 0)
        let lastWithdrawBlock =
valueOrElse(getInteger(toString(i.caller)+"_last_withdraw_block"), 0)

        let previousShare = if(lastDepositBlock < startBlock &&
lastWithdrawBlock < startBlock)
          then {
            let walletBalance =
valueOrElse(getInteger(toString(i.caller)+"_balance"), 0)
            toBigInt(walletBalance) * toBigInt(roundHeight)
          } else {

 parseBigIntValue(valueOrElse(getString(toString(i.caller)+"_"+toString(startBlo
ck)+"_share"), "0") )
          }

        previousShare
      }

      func getTotalPreviousShare() = {
        let lastDepositBlock = valueOrElse(getInteger("last_deposit_block"), 0)
        let lastWithdrawBlock = valueOrElse(getInteger("last_withdraw_block"),
0)

        let previousShare = if(lastDepositBlock < startBlock &&
lastWithdrawBlock < startBlock)
          then {
            let totalBalance = valueOrElse(getInteger("total_balance"), 0)
            toBigInt(totalBalance) * toBigInt(roundHeight)
          } else {

 parseBigIntValue(valueOrElse(getString(toString(startBlock)+"_total_share"),
"0"))
          }

        previousShare
      }

      let walletBalance = walletPreviousBalance - amount
      let withdrawShare = if(height > (startBlock + roundHeight))
        then {
          toBigInt(0)
        } else {
          toBigInt(amount) * toBigInt((startBlock + roundHeight) - height)
        }
      let walletShare = getWalletPreviousShare() - withdrawShare

      let totalBalance = valueOrElse(getInteger("total_balance"), 0) - amount
      let totalShare = getTotalPreviousShare() - withdrawShare
```

```
      let previousWalletActivity = match
(getString(toString(i.caller)+"_activity")) {
        case p: String => true
        case n: Unit => false
      }
      let walletActivity = if(previousWalletActivity)
        then {
          if(!contains(getStringValue(toString(i.caller)+"_activity"),
toString(startBlock)))
            then {
              getStringValue(toString(i.caller)+"_activity") + "_" +
toString(startBlock)
            } else {
              getStringValue(toString(i.caller)+"_activity")
            }
        } else {
          toString(startBlock)
        }

      let isEnded = valueOrElse(getBoolean("ended"), false)

      if(!isEnded)
        then {
          strict withdrawCall = invoke(proxyAddress, "unstakeLP", [amount], nil)

          (
            [
              ScriptTransfer(i.caller, amount, lpAssetId),
              IntegerEntry(toString(i.caller)+"_balance", walletBalance),
              StringEntry(toString(i.caller)+"_"+toString(startBlock)+"_share",
toString(walletShare)),

 IntegerEntry(toString(i.caller)+"_"+toString(startBlock)+"_balance",
walletBalance),
              StringEntry(toString(i.caller)+"_activity", walletActivity),
              IntegerEntry(toString(i.caller)+"_last_withdraw_block",
startBlock),
              IntegerEntry("total_balance", totalBalance),
              StringEntry(toString(startBlock)+"_total_share",
toString(totalShare)),
              IntegerEntry("last_withdraw_block", startBlock)
            ],
            unit
          )
        } else {
          let endBlock = getIntegerValue("end_block")
          (
            [
              ScriptTransfer(i.caller, amount, lpAssetId),
              IntegerEntry(toString(i.caller)+"_balance", walletBalance),
              IntegerEntry(toString(i.caller)+"_last_withdraw_block", endBlock),
              IntegerEntry("total_balance", totalBalance),
              IntegerEntry("last_withdraw_block", endBlock)
            ],
            unit
          )
        }
```

```
    } else {
      throw("")
    }
}

@Callable(i)
func end() = {
  let isEndLocked = valueOrElse(getBoolean("end_locked"), false)
  let startBlock = getIntegerValue("start_block")
  let isEnded =
valueOrElse(getBoolean("start_block_"+toString(startBlock)+"_ended"), false)
  let isValidBlock = height >= startBlock + roundHeight
  if(!isEndLocked && !isEnded && isValidBlock && startBlock < targetBlock)
    then {
      strict previousWXAmount = assetBalance(this, wxAssetId)
      strict previousWXBAmount = assetBalance(this, wxbAssetId)
      strict claimWXCall = invoke(proxyAddress, "claim", nil , nil)
      strict roundWXRewardAmount = assetBalance(this, wxAssetId) -
previousWXAmount
      strict roundWXBRewardAmount = assetBalance(this, wxbAssetId) -
previousWXBAmount

      let totalWXRewardAmount = valueOrElse(getInteger(this,
"total_wx_reward_amount"), 0) + roundWXRewardAmount
      let totalWXBRewardAmount = valueOrElse(getInteger(this,
"total_wxb_reward_amount"), 0) + roundWXBRewardAmount

      let roundShare = match (getString(toString(startBlock)+"_total_share")) {
        case p: String => p
        case n: Unit =>
toString(toBigInt(valueOrElse(getInteger("total_balance"), 0)) *
toBigInt(roundHeight))
      }

      let roundActivity = match (getString("round_activity")) {
        case p: String => p + "_" + toString(startBlock)
        case n: Unit => toString(startBlock)
      }

      let rewardAmount = getIntegerValue("reward_amount")

      if(height >= targetBlock)
        then {
          let totalBalance = valueOrElse(getInteger("total_balance"), 0)
          strict withdrawAll = invoke(proxyAddress, "unstakeLP", [totalBalance],
[])

          (
            [
              ScriptTransfer(i.caller, rewardAmount, wxbAssetId),
              IntegerEntry("start_block_"+toString(startBlock)+"_end_block",
height),
              BooleanEntry("start_block_"+toString(startBlock)+"_ended", true),

 IntegerEntry("start_block_"+toString(startBlock)+"_wx_reward_amount",
roundWXRewardAmount),
```

```
 IntegerEntry("start_block_"+toString(startBlock)+"_wxb_reward_amount",
roundWXBRewardAmount),
              IntegerEntry("total_wx_reward_amount", totalWXRewardAmount),
              IntegerEntry("total_wxb_reward_amount", totalWXBRewardAmount),
              StringEntry(toString(startBlock)+"_total_share", roundShare),
              StringEntry("round_activity", roundActivity),
              BooleanEntry("ended", true),
              IntegerEntry("end_block", height)
            ],
            unit
          )
        } else {
          (
            [
              ScriptTransfer(i.caller, rewardAmount, wxbAssetId),
              IntegerEntry("start_block_"+toString(startBlock)+"_end_block",
height),
              BooleanEntry("start_block_"+toString(startBlock)+"_ended", true),

 IntegerEntry("start_block_"+toString(startBlock)+"_wx_reward_amount",
roundWXRewardAmount),

 IntegerEntry("start_block_"+toString(startBlock)+"_wxb_reward_amount",
roundWXBRewardAmount),
              IntegerEntry("total_wx_reward_amount", totalWXRewardAmount),
              IntegerEntry("total_wxb_reward_amount", totalWXBRewardAmount),
              StringEntry(toString(startBlock)+"_total_share", roundShare),
              StringEntry("round_activity", roundActivity),
              IntegerEntry("start_block", height)
            ],
            unit
          )
        }
    } else {
      throw("")
    }
}

@Callable(i)
func claim(startBlock: Int) = {
  let isClaimLocked = valueOrElse(getBoolean("claim_locked"), false)
  let isEnded =
valueOrElse(getBoolean("start_block_"+toString(startBlock)+"_ended"), false)
  let isClaimed =
valueOrElse(getBoolean(toString(i.caller)+"_"+toString(startBlock)+"_claimed"),
false)
  let firstBlock = getIntegerValue("first_block")

  if(!isClaimLocked && isEnded && !isClaimed && startBlock < targetBlock &&
startBlock >= firstBlock)
    then {
      func getWalletShare() = {
        let lastDepositBlock =
valueOrElse(getInteger(toString(i.caller)+"_last_deposit_block"), 0)
        let lastWithdrawBlock =
valueOrElse(getInteger(toString(i.caller)+"_last_withdraw_block"), 0)
```

```
        let walletShare = if(lastDepositBlock == 0)
          then {
            throw("")
          } else {

 if(valueOrElse(getString(toString(i.caller)+"_"+toString(startBlock)+"_share"),
"0") != "0")
              then {

 parseBigIntValue(getStringValue(toString(i.caller)+"_"+toString(startBlock)+"_s
hare"))
              } else {

 if(valueOrElse(getString(toString(i.caller)+"_"+toString(startBlock)+"_share"),
"-1") == "0")
                  then {
                    toBigInt(0)
                  } else {
                      if(lastDepositBlock < startBlock && lastWithdrawBlock <
startBlock)
                        then {
                          let walletBalance =
valueOrElse(getInteger(toString(i.caller)+"_balance"), 0)
                          toBigInt(walletBalance) * toBigInt(roundHeight)
                        } else {
                          let firstDeposit =
getIntegerValue(toString(i.caller)+"_first_deposit")
                          if(!(firstDeposit < startBlock))
                            then {
                              throw("")
                            } else {
                              let rounds = getStringValue("round_activity")
                              let previousRounds = dropRight(rounds,
(size(rounds) + 1) - value(indexOf(rounds, toString(startBlock))))
                              let theIndexOfTheClosestRound =
if(valueOrElse(lastIndexOf(previousRounds, "_"), 0) != 0)
                                then {
                                  value(lastIndexOf(previousRounds, "_")) + 1
                                } else {
                                  0
                                }
                              let closestRound = drop(previousRounds,
theIndexOfTheClosestRound)

                              let activities =
split(getStringValue(toString(i.caller)+"_activity"), "_")

                              func findClosest(acc: String, next: String) = {
                                let closest = if(value(parseInt(next)) <
value(parseInt(closestRound)) && value(parseInt(next)) > value(parseInt(acc)))
                                    then {
                                      next
                                    } else {
                                      acc
                                    }

                                closest
                              }
```

```
                                let closestActivity =
if(containsElement(activities, closestRound))
                                  then {
                                    closestRound
                                  } else {
                                    let closest = FOLD<53>(activities, "0",
findClosest)
                                    closest
                                  }

                                let theShare =
toBigInt(getIntegerValue(toString(i.caller)+"_"+closestActivity+"_balance")) *
toBigInt(roundHeight)

                                theShare
                              }
                            }
                          }
                        }
                      }

        walletShare
      }

      let walletShare = getWalletShare()

      if(walletShare > toBigInt(0))
        then {
          let totalShare =
parseBigIntValue(getStringValue(toString(startBlock)+"_total_share"))

          let roundTotalWXReward =
valueOrElse(getInteger("start_block_"+toString(startBlock)+"_wx_reward_amount"),
0)
          let roundUserWXReward = toInt(fraction(walletShare,
toBigInt(roundTotalWXReward), totalShare))

          let roundTotalWXBReward =
valueOrElse(getInteger("start_block_"+toString(startBlock)+"_wxb_reward_amount")
, 0)
          let roundUserWXBReward = toInt(fraction(walletShare,
toBigInt(roundTotalWXBReward), totalShare))

          let roundTotalClaimedWX =
valueOrElse(getInteger(toString(startBlock)+"_total_claimed_wx_reward_amount"),
0) + roundUserWXReward
          let totalClaimedWX =
valueOrElse(getInteger("total_claimed_wx_reward_amount"), 0) + roundUserWXReward

          let roundTotalClaimedWXB =
valueOrElse(getInteger(toString(startBlock)+"_total_claimed_wxb_reward_amount"),
0) + roundUserWXBReward
          let totalClaimedWXB =
valueOrElse(getInteger("total_claimed_wxb_reward_amount"), 0) +
roundUserWXBReward
```

```
          let userTotalClaimedWX =
valueOrElse(getInteger(toString(i.caller)+"_total_claimed_wx_amount"), 0) +
roundUserWXReward
          let userTotalClaimedWXB =
valueOrElse(getInteger(toString(i.caller)+"_total_claimed_wxb_amount"), 0) +
roundUserWXBReward

          (
            [
              ScriptTransfer(i.caller, roundUserWXReward, wxAssetId),
              ScriptTransfer(i.caller, roundUserWXBReward, wxbAssetId),

 BooleanEntry(toString(i.caller)+"_"+toString(startBlock)+"_claimed", true),
              IntegerEntry(toString(i.caller)+"_total_claimed_wx_amount",
userTotalClaimedWX),

 IntegerEntry(toString(startBlock)+"_total_claimed_wx_reward_amount",
roundTotalClaimedWX),
              IntegerEntry("total_claimed_wx_reward_amount", totalClaimedWX),
              IntegerEntry(toString(i.caller)+"_total_claimed_wxb_amount",
userTotalClaimedWXB),

 IntegerEntry(toString(startBlock)+"_total_claimed_wxb_reward_amount",
roundTotalClaimedWXB),
              IntegerEntry("total_claimed_wxb_reward_amount", totalClaimedWXB)
            ],
            unit
          )
        } else {
          throw("")
        }

    } else {
      throw("")
    }
}

@Callable(i)
func init(startBlock: Int) = {
  let isInitialized = valueOrElse(getBoolean("initialized"), false)
  if(!isInitialized && containsElement([adminPubKey1, adminPubKey2,
adminPubKey3, adminPubKey4], i.callerPublicKey))
    then {
      (
        [
          BooleanEntry("initialized", true),
          IntegerEntry("start_block", startBlock),
          IntegerEntry("first_block", startBlock),
          IntegerEntry("reward_amount", 10000000000)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
```

```
func changeRewardAmount(amount: Int) = {
  if(containsElement([adminPubKey1, adminPubKey2, adminPubKey3, adminPubKey4],
i.callerPublicKey))
    then {
      (
        [
          IntegerEntry("reward_amount", amount)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Callable(i)
func lock(deposit: Boolean, withdraw: Boolean, end: Boolean, claim: Boolean) = {
  if(i.caller == this)
    then {
      (
        [
          BooleanEntry("deposit_locked", deposit),
          BooleanEntry("withdraw_locked", withdraw),
          BooleanEntry("end_locked", end),
          BooleanEntry("claim_locked", claim)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Verifier(tx)
func verify () = {
  let adminPubKey1Signed = if (sigVerify(tx.bodyBytes, tx.proofs[0],
adminPubKey1))
    then 1
    else 0
  let adminPubKey2Signed = if (sigVerify(tx.bodyBytes, tx.proofs[1],
adminPubKey2))
    then 1
    else 0
  let adminPubKey3Signed = if (sigVerify(tx.bodyBytes, tx.proofs[2],
adminPubKey3))
    then 1
    else 0
  let adminPubKey4Signed = if (sigVerify(tx.bodyBytes, tx.proofs[3],
adminPubKey4))
    then 1
    else 0
  ((adminPubKey1Signed + adminPubKey2Signed + adminPubKey3Signed +
adminPubKey4Signed) >= 3)
}
```

giveaway.ride

```
{-# STDLIB_VERSION 5 #-}
{-# CONTENT_TYPE DAPP #-}
{-# SCRIPT_TYPE ACCOUNT #-}

let giveawayAssetId = base58'Atqv59EYzjFGuitKVnMRk6H8FukjoV3ktPorbEys25on'
let dAppAddress = Address(base58'proxyaddress')
let targetBlock = 3397380

let adminPubKey1 = base58'pubkey1'
let adminPubKey2 = base58'pubkey2'
let adminPubKey3 = base58'pubkey3'
let adminPubKey4 = base58'pubkey4'

@Callable(i)
func giveaway() = {
  if(height < targetBlock && size(i.payments) == 1 && i.payments[0].assetId ==
giveawayAssetId && i.payments[0].amount > 0)
    then {
      let totalGiveaway = valueOrElse(getInteger(this, "total_giveaway"), 0) +
i.payments[0].amount
      let walletGiveaway = valueOrElse(getInteger(this,
toString(i.caller)+"_total_giveaway"), 0) + i.payments[0].amount

      strict giveawayCall = invoke(dAppAddress, "giveaway", nil,
[AttachedPayment(giveawayAssetId, i.payments[0].amount)])

      (
        [
          IntegerEntry("total_giveaway", totalGiveaway),
          IntegerEntry(toString(i.caller)+"_total_giveaway", walletGiveaway)
        ],
        unit
      )
    } else {
      throw("")
    }
}

@Verifier(tx)
func verify () = {
  let adminPubKey1Signed = if (sigVerify(tx.bodyBytes, tx.proofs[0],
adminPubKey1))
    then 1
    else 0
  let adminPubKey2Signed = if (sigVerify(tx.bodyBytes, tx.proofs[1],
adminPubKey2))
    then 1
    else 0
  let adminPubKey3Signed = if (sigVerify(tx.bodyBytes, tx.proofs[2],
adminPubKey3))
    then 1
    else 0
  let adminPubKey4Signed = if (sigVerify(tx.bodyBytes, tx.proofs[3],
adminPubKey4))
    then 1
    else 0
```

```
  ((adminPubKey1Signed + adminPubKey2Signed + adminPubKey3Signed +
adminPubKey4Signed) >= 3)
}
```

# Appendix B: SHA-256 Values for Audited Files

```
giveaway.ride:
0x0248fdf662408f29280157d86558d987b64938165c7b7ec0da6200ec8716018d
lp.ride      :
0x7fe246e26317847f8e7645a05f55e356258bcd714858842b67f5a09035ef92d2
proxy.ride   :
0xdc1eaf22c14912214070da2b840eb41bd7f3edb99e808e2a65cb02e65f7f46f0
```

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech