



ANASTASIA LABS

Security Audit Report

Date: May 14, 2024
Project: TripHut
Version 1.0

Contents

Disclosure	1
Disclaimer and Scope	2
Assessment overview	3
Assessment components	4
Executive summary	5
Code base	6
Repository	6
Commit	6
Files audited	6
Severity Classification	7
Finding severity ratings	8
Findings	9
ID-501 Multiple Fundings	10
ID-502 Upgrade Proposal Spending	11
ID-503 Arbitrary Upgrade Minter Policy	12
ID-504 Unspendable UTxO	13
ID-401 Insufficient UTxO Assets	14
ID-301 Token Dust Attack	15
ID-302 Illegitimate Staking Rewards	17
ID-303 Spam Proposals	18
ID-101 Config UTxO Helper	19
ID-102 Misleading Expression Name	20
ID-103 Inefficient Check	21
ID-104 Safe Helper	22
ID-105 Integer To ByteString	23

Disclosure

This document contains proprietary information belonging to Anastasia Labs. Duplication, redistribution, or use, in whole or in part, in any form, requires explicit consent from Anastasia Labs.

Nonetheless, both the customer Yaad Labs and Anastasia Labs are authorized to share this document with the public to demonstrate security compliance and transparency regarding the outcomes of the Protocol.

Disclaimer and Scope

A code review represents a snapshot in time, and the findings and recommendations presented in this report reflect the information gathered during the assessment period. It is important to note that any modifications made outside of this timeframe will not be captured in this report.

While diligent efforts have been made to uncover potential vulnerabilities, it is essential to recognize that this assessment may not uncover all potential security issues in the protocol.

It is imperative to understand that the findings and recommendations provided in this audit report should not be construed as investment advice.

Furthermore, it is strongly recommended that projects consider undergoing multiple independent audits and/or participating in bug bounty programs to increase their protocol security.

Please be aware that the scope of this security audit does not extend to the compiler layer, such as the UPLC code generated by the compiler or any areas beyond the audited code.

The scope of the audit did not include additional creation of unit testing or property-based testing of the contracts.

Assessment overview

From <DATE>, <YEAR> to <DATE>, <YEAR>, Yaad Labs engaged Anastasia Labs to evaluate and conduct a security assessment of its TripHut protocol. All code revision was performed following industry best practices.

Phases of code auditing activities include the following:

- Planning – Customer goals are gathered.
- Discovery – Perform code review to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through testing and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities.

The engineering team has also conducted a comprehensive review of protocol optimization strategies.

Each issue was logged and labeled with its corresponding severity level, making it easier for our audit team to manage and tackle each vulnerability.

Assessment components

Manual revision

Our manual code auditing is focused on a wide range of attack vectors, including but not limited to.

- UTXO Value Size Spam (Token Dust Attack)
- Large Datum or Unbounded Protocol Datum
- EUTXO Concurrency DoS
- Unauthorized Data modification
- Multisig PK Attack
- Infinite Mint
- Incorrect Parameterized Scripts
- Other Redeemer
- Other Token Name
- Arbitrary UTXO Datum
- Unbounded protocol value
- Foreign UTXO tokens
- Double or Multiple satisfaction
- Locked Ada
- Locked non Ada values
- Missing UTXO authentication
- UTXO contention

Executive summary

TripHut is a Decentralised Autonomous Organization built on the Cardano Blockchain designed to send its members on free or discounted vacations using community resources. Awardees of the travel allocation are decided on a relative majority basis, where members submit travel requests, and the applications with the most votes are selected. The smart contract thereafter disperses the funds to member and designated wallets to cover and handle: Airfare, Accommodation, Transportation, Pocket Money and Agent Fees.

Code base

Repository

<https://github.com/yaadlabs/DAO>

Commit

ade0a5a76b156d77e48f10fcc35d752e7c682da6

Files audited






SHA256 Checksum	Files
b06d27d9ea650db5b0404948104e642e345c7a30a4217a7a2ffd1e6adba214e0	dao/dao-lib/Dao/Tally/Script.hs
eb73d7f5499884457a6101c473976fcdef29ef6f3c0b05534e9068567c26b923	dao/dao-lib/Dao/Treasury/Script.hs

Severity Classification

- **Critical:** This vulnerability has the potential to result in significant financial losses to the protocol. They often enable attackers to directly steal assets from contracts or users, or permanently lock funds within the contract.
- **Major:** Can lead to damage to the user or protocol, although the impact may be restricted to specific functionalities or temporal control. Attackers exploiting major vulnerabilities may cause harm or disrupt certain aspects of the protocol.
- **Medium:** May not directly result in financial losses, but they can temporarily impair the protocol's functionality. Examples include susceptibility to front-running attacks, which can undermine the integrity of transactions.
- **Minor:** Minor vulnerabilities do not typically result in financial losses or significant harm to users or the protocol. The attack vector may be inconsequential or the attacker's incentive to exploit it may be minimal.
- **Informational:** These findings do not pose immediate financial risks. These may include protocol optimizations, code style recommendations, alignment with naming conventions, overall contract design suggestions, and documentation discrepancies between the code and protocol specifications.


Finding severity ratings

The following table defines levels of severity and score range that are used throughout the document to assess vulnerability and risk impact.

	Level	Severity	Findings
	5	Critical	4
	4	Major	1
	3	Medium	3
	2	Minor	0
	1	Informational	5

Findings

ID-501 Multiple Fundings

	Level	Severity	Status
	5	Critical	Acknowledged

Description

Given a proposal, of any of the three types, which has sufficient 'for' votes to make it eligible for being funded from the treasury; the Treasury validator allows that proposal to be funded multiple times in subsequent transactions as long as there are enough funds left.

This is made possible because the validator accepts the 'Tally NFT' containing UTxO (Tally UTxO), as a reference input. After the proposal being funded once, the Tally UTxO is still available to be used in a subsequent transaction to receive funding from treasury.


Recommendation

Once a proposal has been funded, it should reflect a change in the state of Tally UTxO to make it ineligible for being used once again. This can be achieved by requiring the Tally UTxO to be spent rather than being referenced. And having the respective 'Tally NFT' to be burnt in that spending transaction which obtains funding from Treasury validator.

Resolution

Pending

ID-502 Upgrade Proposal Spending

	Level	Severity	Status
	5	Critical	Acknowledged

Description

Upgrade proposals are currently allowed to spend a Treasury UTxO and claim all the funds therein. This helps the protocol to transfer the funds to a new Treasury Script if the upgrade demands so. However, this leaves the door open for any upgrade proposal to drain the Treasury funds without the accepted upgrade proposal demanding it.

Also, currently there is no validation if the funds are transferred to a new Treasury script address. An attacker can take advantage of an approved upgrade proposal (which may or may not require funds to be transferred to new Treasury) and be able to claim all the assets locked in the current Treasury UTxO.

Recommendation

It is recommended that the 'Upgrade' data constructor of type 'ProposalType' should contain explicit information regarding whether this proposal needs to send funds to a new treasury address or not. Additionally, have the new script address be provided in it.

ProposalType = Upgrade CurrencySymbol (Maybe Address)


Update treasury script to allow spending of its UTxOs only if there's an address provided ('Just newScriptAddress' and not allow spending if its 'Nothing') and validate that all funds are sent to 'newScriptAddress'.

This approach also facilitates providing voters with verifiable on-chain information of the new treasury script address.

Resolution

Pending

ID-503 Arbitrary Upgrade Minter Policy

	Level	Severity	Status
	5	Critical	Acknowledged

Description

The Treasury validator checks for the presence of a token mint with a specific currency symbol (obtained from within 'TallyStateDatum'proposal' having data constructor 'Upgrade CurrencySymbol') before allowing spending of its UTxO. This check is also performed by 'Configuration/Scripts.hs' in the same transaction before allowing the update of 'DynamicConfigDatum'.

This currency symbol or policy id of Upgrade Minter policy is not checked during the creation of 'TallyStateDatum' in Tally minter 'mkTallyNftMinter'. Allowing for an arbitrary minting policy to be provided in what would look like a legitimate 'TallyStateDatum'. Going against the protocol's intention of it being a one shot minting policy, allowing the mints to be unrestricted. In conjunction with vulnerability 502: Upgrade Proposal Spending, this could lead to spending of all Treasury UTxOs, one in each subsequent transaction. Additionally, it can even lend attacker the ability to update 'DynamicConfigDatum' more than once.


Recommendation

Instead of relying on a separate minting policy to do required validations during update proposal we can use the burning of 'Tally NFT' (already required from recommendation of vulnerability 501: Multiple Fundings). The 'mkTallyNftMinter' can perform the same validations which was supposed to be done in the questionable minting policy, when a 'Tally NFT' belonging to proposal type 'Upgrade' (by checking 'TallyStateDatum' in 'Tally UTxO') is burnt.

Resolution

Pending

ID-504 Unspendable UTxO

	Level	Severity	Status
	5	Critical	Acknowledged

Description

The Treasury Validator ('validateTreasury') does not make use of datum in its validation. So, it also skips checking the datum of output UTxO, with leftover funds, being returned to the script. However, it's essential to have datum in every UTxO locked at a script address, otherwise the UTxO becomes locked forever.

Recommendation


It's highly recommended to check the datum of continuing output UTxO to have a specific datum. Having a specific datum prevents arbitrary datums, of varying sizes, to be set in a UTxO which can lead to unnecessary UTxO bloat and increased transaction costs.

Note: Care must be taken in ensuring that 'generalPaymentAddress', 'travelAgentAddress' and 'travelerAddress' are not script addresses due to lack of datum check for their outputs.

Resolution

Pending

ID-401 Insufficient UTxO Assets

	Level	Severity	Status
	4	Major	Acknowledged

Description

The Treasury validator allows spending of only one UTxO to fund no more than one proposal in a single transaction. This would keep decreasing the assets (ADA for now) held in the UTxO as proposals are funded one after the other. Eventually, the assets would drop to such a threshold value that it won't be enough to fund any proposal completely. The assets as such won't be insignificant in value just that they won't meet the funding requirements of any proposal.

The only option left then would be to create a proposal specifically to spend this amount back to any protocol controlled wallet. However, given that voting itself is a time consuming costly process, it may not be economically feasible to spend the low value locked UTxO. Thereby, leaving that UTxO and assets therein unusable.

Recommendation

An easily implementable recommendation here would be, to check the value of the output being returned to script. If it is less than a predetermined value (could be maintained in 'DynamicConfigDatum'), rather than returning it to Treasury script, it should be sent to a protocol controlled multisig wallet whose address is also present in 'DynamicConfigDatum'. This allows these low value UTxOs to be used by the protocol in future.

A more powerful and effective, albeit difficult to implement recommendation would be to allow spending multiple script UTxOs in one transaction. This should only be allowed by implementing the Withdraw Zero Trick. This design pattern would allow the Treasury script to validate the transaction correctly even with multiple of its inputs being used without the risk of 'Double Satisfaction Attack'. Low value UTxOs can be easily used as inputs and be sent back as a single UTxO to contract minus the funding amount.

Resolution

Pending

ID-301 Token Dust Attack

	Level	Severity	Status
	3	Medium	Acknowledged

Description

Values of all UTxOs sent to a script address must have an upper bound for their size, and the upper bound should be low enough to not prevent consumption of the UTxO as an input in a future transaction. If this isn't taken care of, a script UTxO can be subject to being filled with many random tokens which can increase the transaction fees of subsequent transactions. It can also make the script UTxO unspendable in cases where new token(s) need to be added to it.

This common vulnerability is known as "Token Dust Attack." We found this attack vector at the following places.

In 'Treasury/Script.hs' at:

1. Line 283

```
outputValueIsLargeEnough :: Bool
!outputValueIsLargeEnough = outputValue 'geq' (inputValue -
    disbursedAmount)
```

While the above ">=" check confirms that the desired value is returned back to the script, it falls short to check that there aren't unnecessary tokens present in 'output-Value'.

2. Line 288 There could be an instance of this vulnerability if 'travelAgentAddress' happened to be a script address. The address would be sent a UTxO with the right lovelaces in addition to many random tokens in it.
3. Line 292 There could be an instance of this vulnerability if 'travelerAddress' happened to be a script address. The address would be sent a UTxO with the right lovelaces in addition to many random tokens in it.
4. The same holds when the 'ProposalType' is 'General' at Line 326 and 331.

In 'Tally/Script.hs' at:

1. Line 456
2. Line 434

Recommendation


The recommendation here is to check that UTxO's value consists the exact number of required assets in addition to ">=" check being performed currently. i.e 1 asset, ADA, for

all the instances in 'Treasury/Script.hs' and 2 assets, ADA and Tally NFT/ Vote NFT, for the instance in 'Tally/Script.hs'.

Resolution

Pending

ID-302 Illegitimate Staking Rewards

	Level	Severity	Status
	3	Medium	Acknowledged

Description

Whenever a validator needs to check where a UTxO is coming from (or going to), an important consideration is the staking part of the address. If only the payment part of the address is checked, it can potentially allow the staking part of the UTxOs to be changed arbitrarily thereby taking control of staking rewards.

We spotted the occurrences at:

1. 'outputValue' at 'Treasury/Script.hs' does not check the staking part of the continuing output at Line 278 and Line 321.
2. 'outputOnTallyValidator' 'newValue' at 'Tally/Script.hs' do not check the staking part of the continuing output at Line 212 and Line 448. However, this seems quite minor as the locked ADA would be small.

Recommendation

In all cases, consider implementing validations for staking parts too i.e. instead of just checking the payment part, check the whole script address.

Resolution

Pending

ID-303 Spam Proposals

	Level	Severity	Status
	3	Medium	Acknowledged

Description

The minting policy 'mkTallyNftMinter' which is responsible for creating a 'Tally NFT' for a given index (proposal number), allows anybody to mint tally NFTs with incremental indices. This could lead to creation of multiple junk UTxOs with Tally NFTs residing at Tally validator address. It also bloats up the index value in 'IndexDatum' (datum of UTxO with 'Index NFT' that keeps count of the next proposal number to be created).

The above issue on its own can be considered harmless with mentioned inconveniences. However, if these spam proposals stand to get the required number of votes, either by misleading honest voters or by accepting votes from malicious users, then treasury funds can be compromised.

Recommendation


Instead of being open for all, the minting policy can implement a multisig contract to allow only the core members of the protocol to mint 'Tally NFTs' or require something similar to vote pass logic, required for voting, to be implemented so that only members of the DAO can create 'Tally NFTs'/proposals.

Note: The DAO seems to implement members-only voting by means of vote pass, 'dynamicConfigDatum'voteNft'. Since Vote Minter Validator are out of this audit's purview, we recommend to ensure its full-proof functioning so that non-members don't make use of spam proposals to drain treasury.

Resolution

Pending

ID-101 Config UTxO Helper

	Level	Severity	Status
	1	Informational	Acknowledged

Description

We found the below snippet of code at five different validators.

```
-- Helper for filtering for config UTxO in the reference inputs
hasConfigurationNft :: Value -> Bool
hasConfigurationNft = hasOneOfToken configSymbol configTokenName

-- Get the configuration from the reference inputs
DynamicConfigDatum {dynamicConfigDatum'tallyValidator} =
case filter (hasConfigurationNft . txOutValue . txInInfoResolved)
  txInfoReferenceInputs of
    [TxInInfo {txInInfoResolved = TxOut {..}}] -> convertDatum txInfoData
      txOutDatum
    _ -> traceError "Should be exactly one valid config in the reference
      inputs"
```

Recommendation

Abstracting this logic into a separate function would result in a cleaner, more maintainable and consequently safer code.

Resolution

Pending

ID-102 Misleading Expression Name

	Level	Severity	Status
	1	Informational	Acknowledged

Description

In 'Tally/Script.hs' at Line 440

```
tallyingIsInactive :: Bool  
!tallyingIsInactive = tallyStateDatum 'proposalEndTime' 'before' '  
    txInfoValidRange
```

The expression name is misleading as it evaluates to 'tallyingIsActive' and not 'tallyingIsInactive'.

Recommendation

Update expression name.

Resolution

Pending

ID-103 Inefficient Check

	Level	Severity	Status
	1	Informational	Acknowledged

Description

In 'Tally/Script.hs' at Line 443

```
voteTokenAreAllBurned :: Bool
!voteTokenAreAllBurned = not $ any (hasVoteWitness . txOutValue)
    txInfoOutputs
```

The above function has to iterate over all the tx outputs for a successful validation, which is very costly and can be avoided.

Recommendation


Instead of the above, an existing computation in the script can be modified to get the total number of vote witnesses in the inputs. What remains next is to just check that 'txInfoMint' contains burning of equal number of vote witnesses.

The computation referred to is stepVotes. In which instead of 'hasVoteWitness' a helper like 'countVoteWitness' could be ran over 'txOutValue' and its value be accumulated over all inputs.

Resolution

Pending

ID-104 Safe Helper

	Level	Severity	Status
	1	Informational	Acknowledged

Description

We see many occurrences of 'hasOneOfToken' function used to confirm that value has atleast one of given token. This however is being done for tokens which should be NFTs and so shouldn't be having more than one quantity.


Recommendation

Its recommended to check the same condition for NFTs using 'hasSingleTokenWithSymbolAndTokenName', which fails if there isn't exactly one token in the UTxO value.

Resolution

Pending

ID-105 Integer To ByteString

	Level	Severity	Status
	1	Informational	Acknowledged

Description

In 'Shared.hs' at Line 164

```
integerToByteString :: Integer -> BuiltinByteString
integerToByteString n
```

The above function allows negative integers and tries to convert it into bytestring incorrectly.

Recommendation

Since the intended purpose of the function is to just convert non-negative integers to bytestring it should fail when given a negative integer value. While the case for negative integers might never happen in the Index Datum's index, its still safer to handle negative integer values with an error.

Resolution

Pending