MuesliSwap Onchain Governance Audit Report

James Koppel, Mirdin, LLC | April 17, 2024

Executive Summary and Scope

MuesliSwap is a hybrid orderbook and automated market-maker DEX on Cardano. Liquidity providers on MuesliSwap pools receive governance token. MuesliSwap Onchain Governance is a protocol by which owners of MuesliSwap governance tokens may vote on policies, including distributing treasury funds, authorizing actions in not-yet-written contracts, and migrating the governance protocol to new contracts. Because a separate mechanism, outside the scope of this audit, allows governance token holders to stake their tokens in a vault in exchange for rewards, this protocol allows users to vote either with their governance token directly or with a new fungible token representing staked governance token. There is also a mechanism by which users who staked governance token prior to the creation of this protocol may, with the authorization of the MuesliSwap protocol owners, retroactively mint token representing their staked position up to some predetermined global cutoff.

All contracts of the protocol are written in OpShin, a Python-based smart contract language for Cardano created by Niels Mündler.

This audit and report was completed by Mirdin Verification, the crypto-auditing arm of Mirdin, the Code Quality Company.

Scope and Methodology

Mirdin performed a de novo analysis of the smart contracts providing the MuesliSwap Onchain Governance Protocol. Prior to the audit, Mirdin performed scoping, and requested the MuesliSwap team document the interactions between the many contracts of the protocol. Upon beginning the audit, Mirdin repeatedly manually inspected the code contained in the respective files and attempted to locate potential problems in one of these categories:

a) Errors that allow undue seizure or locking of user or treasury tokens
b) Errors that allow manipulation of governance actions
c) Denial of service of legitimate tally or governance actions

The OpShin compiler and libraries were explicitly out of scope. Mirdin did nonetheless inspect a few library implementations during the course of the audit.

Although not explicitly in scope, we also opportunistically recorded gas and code quality issues.

Disclaimer

This report is presented without warranty or guarantee of any type. This report represents a best-effort approach to finding undesired behavior and bugs

within the time alotted. Mirdin advises against making any business or other decisions based on this report at this time. The focus of this report is on the technical implementation provided by the contractors and subcontractors for OpenSwap, based on the information provided by these parties, and does not assess the concept, mathematical validity, or business validity of OpenSwap's product. Additionally, this report does not address the financial viability or suitability of the implementation for any purpose.

Audit

Methodology

Mirdin was initally hired to audit the staking and treasury contracts of MuesliSwap Onchain Governance, as these contracts directly control funds and governance tokens. However, auditing these contracts also required significant reading of the Tally contract, which controls the authorization of many treasury and staking actions, whose validators are intended to run concurrently with those of the Staking contract. After seeing that Mirdin Verification had already performed substantial reading of the Tally contract and was opportunistically reporting issues, the MuesliSwap team decided to extend the audit to cover the entirety of MuesliSwap Onchain Governance. All auditing was performed on commit `d0f20afb5f1f395cb75edc4d8c` of the repository[1] . Mirdin did not perform any dynamic testing of the protocol.

Files inspected

These are the files inspected, as of commit `d0f20afb5f1f395cb75edc4d8c`.

All git hashes were computed with `git hash-object`.

All line counts were computed with `wc -l`.

1 github.com/muesliswaplabs/muesliswap-onchain-governance>, accessed March 17, 2024

git hash	File Name	LOC
76523402ae	one_shot_nft.py	38
1da2200545	util.py	422
b9227561d6	gov_state/gov_state.py	317
dad7ddebb5	gov_state/gov_state_nft.py	14
1a5501e19f	licenses/licenses.py	124
b6d0189538	staking/staking.py	210
495b7eedce	staking/staking_util.py	188
6dd0fd324e	staking/staking_vote_nft.py	110
049e3513f4	vault/vault_ft.py	141
c8156527b6	vault/vote_permission_nft.py	45
442ef10f9c	tally/tally.py	377
58714ab3be	tally/tally_auth_nft.py	56
950f0cf2c5	treasury/treasurer.py	266
d9bad4c51d	treasury/treasurer_nft.py	13
019197ce3d	treasury/util.py	25
9af50c1b01	treasury/value_store.py	55
ad1d9eb50f	utils/ext_interval.py	64

Findings

H-001: Delegated voters may add/retract votes in a cycle, draining a staking position of all Ada

The Staking module contains a mechanism allowing a staking position owner to delegate the actual voting in a tally, reducing contention on the Tally UTXO. This mechanism consists of attaching an NFT committing to a DelegatedAddVote or DelegatedRetractVote redeemer to the staking position. However, by default, these NFTs are not burned and may be reused.

Thus, if a user votes and then retracts a vote, and the NFTs associated with both the DelegatedAddVote and DelegatedRetractVote actions remain attached to the staking UTXO, an adversary may repeatedly add and retract votes in an endless cycle.

For each such action, the contract is designed to pay the delegatee up to 2 Ada as payment for performing the action. Thus, by repeating such actions in a cycle, the adversary may completely drain the staking position. If a staking position is overfunded and transaction fees drop low enough, this can be a profitable activity for the attacker.

Recommendation

Ensure all vote permission NFTs are burned on use.

H-002: Users may record a vote in a staking position without actually voting, causing multiple issues

Each vote in a tally is recorded in three places. First, the tally state records the number of votes for each proposal. Second, each staking position records each tally it has voted in, along with authorizing information and the weight of the vote. Third, an NFT is minted recording the vote. These three actions are intended to take place in the same transaction. To retract a vote, all three must be done in concert.

However, it is possible to do the second of these actions without the first, because the staking contract never checks for either the participation of the tally contract nor the vote-record-NFT minting policy.

Multiple issues stem from this. For example, it is possible for an adversary to, when the owner delegates a vote, record the vote in the staking position but not the tally contract. This causes the staking position owner to effectively not vote for the tally, but instead prevents them from doing so. They will not be able to retract the vote at all (due to the absence of vote-record NFT), and the

governance tokens intended to vote for the tally will be locked on the staking position until the end of the tally, or forever if the tally has no end date.

A more minor issue is that, if an adversary somehow gets ahold of the vote-record NFT (say, by the staking position owner removing it and then sending it somewhere), then it is possible for them to perform the `RetractTallyVote` action in the tally contract but the `AddVote` action in the staking contract.

Recommendation

Require that the tally contract validate each transaction where a vote is recorded in a staking position, which in turns requires that the staking vote NFT is correctly minted.

H-003: Lack of checks on the variant of a tally winner payload permits punning, potential seizure of protocol

Tally proposals each contain a payload datum indicating what should happen if that proposal wins the most votes. The current governance code contains three variants of the this payload datum: `GovStateUpdateParams`, `LicenseReleaseParams`, and `FundPayoutParams`. These are distinguished by their `CONSTR_ID`. The MuesliSwap

Here is the worst-case scenario:

1. A developer adds a new kind of contract to the governance protocol on which tallies can be voted. This contract is programmed so that certain winning tallies will authorize it to do something. Call the corresponding variant of the tally winner payload `DoSomethingUserParams`.
2. However, it so happens that the `DoSomethingUsefulParams` has the same data layout as either `GovStateUpdateParams` or `FundPayoutParams`.
3. The developer begins an unobjectionable tally, and a winning proposal is chosen
4. The developer then invokes the governance or treasury contracts passing in this tally. Said contract treats the tally winner payload as an instance (respectively) of `GovStateUpdateParams` or `FundPayoutParams`, and either hands control of the entire protocol to the developer, or pays out the entire treasury.

Although difficult to pull off, this attack is catastrophic enough to merit a High rating.

A more minor concern is that there must be a global registry of all `CONSTR_ID` values for tally winner payload variants. Replacing the `CONSTR_ID` values with hashes of the data structure mitigates this concern greatly.

Recommendation

Check the `CONSTR_ID` (e.g.: using an OpShin `instanceof` check) at all use-sites of a tally proposal payload datum.

H-004: Any user or coalition who temporarily acquires a quorum may seize 100% control of the treasury and governance

Nothing prevents a tally from being created where all proposals are identical. In fact, nothing prevents a tally from being created that only has a single proposal. If a group controls a quorum of the governance token, then they can create a tally whose sole proposal grants them full control of the governance protocol and/or grants them all funds from the treasury. Then they may simply vote for this proposal. Even if 100% of other token holders oppose, there is nothing they can do: the proposal will pass in the minimum allowed tally time, and no countering governance action may be taken in that time period.

The MuesliSwap team stated they were considering a value of quorum equal to "30-40% of the total supply," permitting a group with as low as 31% of the governance token to seize total control.

Potential fixes to M-004 may exacerbate this issue.

Recommendation

Require all tallies have a default “do nothing” option

H-005: A tally auth NFT may be minted and sent to any address whenever the governance parameters change, allowing users to create an authorized tally with arbitrary parameters

The `tally_auth_nft` minting policy allows minting a tally auth NFT in any transaction validated by the governance contract. When an `UpgradeGovState` action is performed on the governance contract, there is a check that nothing is sent to the tally address, but nothing prevents a tally auth NFT from being minted to an arbitrary different address.

Thus, whenever a proposal passes to do an `UpgradeGovState`, whichever user executes the action may mint a tally auth NFT under their control. They may then attach it to a new tally datum sent to the tally contract, with arbitrary parameters. All other contracts will respect this tally as being authorized by the governance thread. A user may manipulate the parameters on this tally as they wish: by setting the end date in the past, by setting the quorum to 0, by setting any number of votes as already being recorded, by setting any proposals they wish, and by setting the proposal ID arbitrarily high. They may thus immediately take control of the governance contract, pay out the entire

treasury, and also set the `last_applied_proposal_id` of the treasurer or governance state to an arbitrarily high value that prevents further proposals from being applied.

Recommendation

Don't allow this minting to occur.

H-006: Governance update cannot be applied if anyone has created a tally since the creation of the governance-update tally

Consider this code in `validate_update_gov_state` of `gov_state.py`

```
    tally_result = winning_tally_result(
        redeemer.tally_input_index,
        Token(params.tally_auth_nft_policy,
↪   params.gov_state_nft.token_name),
        tx_info,
        state.last_proposal_id, # This is wrong
```

```
        True,
    )
```

`winning_tally_result` is designed to error if the tally being checked has an ID greater than the 4th argument. In this case, that is `state.last_proposal_id`, which can be incremented by anyone at any time by creating a new tally. Therefore, a governance update cannot be applied if anyone has created a tally since the creation of the governance-update tally. Ergo, anyone can prevent any governance update from occurring.

Recommendation

Change this call to use `latest_applied_proposal_id` instead.

M-001: Size checks of staking positions too tight: prevents valid participation, enables temporary malicious bricking

UTXOs in the main Staking contract are limited by explicit checks to have at most 1000 bytes each in the datum and the value. This value is too small; each vote in a tally is estimated to require 80 bytes, and the staking params are estimated to required 200 bytes.

Further, delegated users of a staking position may add dust to bring it near the maximum size and prevent addition of further (staked) governance token. However, this is a very minor issue, as the owner can always remove and consolidate tokens on the contract.

Recommendation

Increase allowed size for this contract

M-002: Incorrectly-implemented check for whether enough governance token has been staked

The `check_enough_governance_tokens_in_output` check in `staking.py` is intended to ensure that, if a user is voting with weight W in a tally that ends at time T, then the user will have at least W governance tokens attached to the staking position at time T. This includes governance tokens directly attached to the staking position, and also governance tokens which are staked in the vault until at least time T.

However, this check is incorrectly written, and compares the weight voted in each tally with the amount of token staked until whichever tally has the latest ending time. In the case the user has voted, even with weight 1, in a tally that

has no ending time, then no staked token will be counted towards any other tallies the user has voted in.

Recommendation

Correct the `check_enough_governance_tokens_in_output` logic.

M-003: Incorrect value-preservation check means delegated voters can never retract a vote

The staking contract checks that a delegated voter does not remove any token from a staking position except for their payment of up to 2 Ada. Retracting a vote from the staking contract requires the transaction be validated by the tally contract. The tally contract requires that the vote-record NFT be burned in order to retract a vote. However, as the vote-record NFT is minted to the staking position, the previous check prevents the vote-record NFT from being burned.

Adding these together, a delegated non-owning voter cannot retract a vote, unless the owner removes the NFT from the staking position and otherwise places it under control of the delagatee.

Recommendation

Fix this check to allow for burning the vote-record NFT

M-004: Nonstandard definition of "quorum" means that tallies with many proposals may have no winner

"Quorum" is defined "the number of members of a group or organization required to be present to transact business legally, usually a majority." In a blockchain governance context, this would correspond to the amount of governance token that must be staked in a tally for a winner to be determined.

However, the "quorum" check in this protocol instead checks that the amount of votes for the winning proposal must be of at least a certain quantity, erroneously called the "quorum."

Negative consequences include: if there is a tally with many proposals, it is possible to end up in a situation where all governance token in existence has participated in the tally, but no winner is found.

Recommendation

Change to a conventional quorum system, requiring a certain number of total votes. Optionally add a second number for the fraction or quantity of votes that a proposal must have to be deemed the winner.

L-001: Treasury payouts may have double-satisfaction with other payouts to same address

If a payout is to be sent to an address, and some other protocol also has a prize to be sent to the same address at the same time, then the same payment may satisfy both.

This can also occur if anyone has any need to pay for anything else to an address at around the same time that a tally passes to pay this address..

Recommendation

Do not allow treasury payouts to occur when any other script is running in the transaction.

L-002: Executing a later tally winner prevents earlier tally winners from being executed, even if not already executed

The treasury contract tracks the proposal ID of the last applied payout, and will not perform any actions for winning proposals with earlier IDs.

This means that, if tally 5 and tally 6 conclude around the same time, and the payout for tally 6 is executed first, then the payout for tally 5 can never be executed.

Recommendation

The MuesliSwap team believes this is acceptable, and that it is not intended that there will be tallies that can terminate around the same time. In case there are, they say the first tally should simply be repeated.

L-003: Max tally ID in the license contract is too low

License NFTs must have the first 3 bytes of the token name equal to the tally ID that authorized their creation. If more than $2^{24} < 17M$ tallies are created, then no more license NFTs may be minted.

At present, the throughput of the Cardano change is too low to make this conceivable. But for comparison, the Solana blockchain is theoretically capabale

of executing over 100,000 transactions every few seconds, and thus would allow the creation of 17M tallies in under an hour. Because tally creation is unpermissioned, if the Cardano blockchain has its throughput increased to rival that of Solana, and especially if transaction fees fall a corresponding amount, then an adversary can prevent the creation of all future licenses by creating 17M worthless tallies.

Recommendation

Increase the number of bytes allocated to the tally ID in the license NFT names. It correctly allocates 29 bytes to the timestamp, which is far more than sufficient.

L-004: Quadratic algorithm may allow adversaries to lock funds in the treasury

The function `merge_without_duplicates`, called on a map with M keys and a map with N keys, has complexity O(MN). This creates particular risk in `check_fund_distribution_correct` in `treasurer.py`, where multiple transitive callers of this function run, and slightly more computation runs on the old state than the new state. The risk is that someone will be able to add dust to a UTXO in the value store in a manner where the validation script

comes up just under the gas limit, but attempting to remove it goes over the gas limit. This effectively allows anyone to run the `ConsolidateFunds` action to tamper with any existing UTXO in the value store in a manner that freezes it in the value store.

We have not created a POC verifying this attack can be carried out, and it's possible that the max value size is low enough that it is not possible to cause out-of-gas. We nonetheless include it without verifying because, if this attack is absent, then its absence is very fragile, and very minor code changes can put the time of this contract above the threshold.

Recommendation

In the treasury contract, limit the number of token types that may be attached to an outgoing UTXO sent to the value store.

Code quality issues

- `staking.py`, function `check_extract_max_2_ada`, has a variable incorrectly named `prev_value_minus_one_ada`
- The `check_mint_exactly_one_with_name` function and friends suggest that they only check that a certain amount of token was minted. The name should be updated to reflect that they also check that no other token from the same minting policy is minted. This is an instance of the

Unexpected Side Effect Linguistic Anti-Pattern (https://www.linguistic-antipatterns.com)

- `vote_permission_nft.py` gives type signatures requiring a `VotePermissionNFTParams`. However, it's intended that, if the contract is being used to burn, no valid `VotePermissionNFTParams` will be provided. Recommend changing the input type to `Union[None, VotePermissionNFTParams]` to communicate this.
- `check_payout_executed_correctly` in `treasurer.py` is misnamed. It computes the correct payout, and does some other checks, but does not check if the correct payout was executed.
- The docstring for `check_preserves value` states "Check that the value of the previous state input is equal to the value of the next state output." It actually checks for greater-than-or-equal.

Gas issues

- The loop in `treasurer.py` which repeatedly invokes the function `check_output_reasonably_sized` includes many redundant checks that the same datum is reasonably sized
- `staking_vote_nft.py` runs `ote_index = redeemer.vote_index` and then accesses `reedemer.vote_index` again. Presumably this compiles into redundant lookups.
- The pattern `len([i for i in some_list if some_condition])` occurs many times. Unless a compiler optimization fixes this, this wastefully performs two passes over the list
- `check_valid_staking_update` in `tally.py` calls the function `translate_to_participation` even if the result is not used
- `remove_participation_at_index` in `utils.py` computes `list[:index] + list[index + 1 :]`, which does two passes

over the list. (Hopefully the latter slice is compiled to merely take the suffix of the list and not compute a new one.)
- In `winning_tally_result` in `utils.py`, the call to `list_index` does a second pass over the list, redundant with the call to `max`

Other notes

- The `vault_ft` contract assumes the vault owner is trusted; if the vault owner authorizes it, the `vault_ft` code theoretically makes it possible to mint token for a position that is simultaneously being withdrawn.

- There is no mechanism to premptively restake any governance token whose stake expires before some critical tally deadline. Consider: Suppose Bob has 1 million governance token staked until Tuesday. Then a tally appears, with vote expiring Wednesday. Bob won't be able to vote until Tuesday, when he withdraws and then re-stakes all his token.

- A comment in `validate_new_tally` in `gov_state.py` describes a check as "ensure that the new tally has an auth nft (and only one)." However, it does not check for only one being present, although it is already the case that only one can be minted.

- We have recommended that the token names of license NFTs allocate some bytes to the hash of `winning_tally_params.datum`, and that doing so would reduce the attack surface in contracts consuming license NFTs. The MuesliSwap team disagreed that this is significant.

- The validator in `licenses.py` contains a loop intended to execute ex-

actly once. This code is strange; unsure how the gas usage compares to not having a loop.

- Strictly speaking, the vote-permission NFT is not an NFT, because there can be multiple.

Conclusion

This document outlines the 14 concerns (above a minimal threshold of significance) we found when reviewing the MuesliSwap Onchain Governance Protocol, along with numerous minor code quality and gas comments. The sheer complexity of this protocol, which frequently requires multiple validators to run in a single transaction, allows for numerous attacks in even defensively-written code. Fortunately, none of these issues are particularly deep or hard to patch.

As stated earlier in this report, Mirdin does not recommend for nor against the use of any work referenced in this report, and makes no guarantee of any kind regarding the contents or exhaustiveness of this report.

Fixed versions

All issues have been fixed as of commit `d268cba9f5`, with the exception of L-002, where the risk was previously known to and accepted by the MuesliSwap team.

- **H-001**: Fixed in `ff3c16e` and `f711886` by requiring that vote permission NFT be appropriately burned.
- **H-002**: Fixed in `93b5381` by checking that vote-record NFT be minted when running the `AddVote` action on a staking position
- **H-003**: Fixed in `3e4bb3f` by inserting calls to `check_integrity` at all use-sites of tally winner payloads
- **H-004**: Fixed in `7305062` by requiring that all tallies have a no-op operation as one option
- **H-005**: Fixed in `0444a5f` by checking that no tally auth token is minted during an `UpgradeGovState` action
- **H-006**: Fixed in `11800cb` by fixing the relevant parameter
- **M-001**: Fixed in `5df26ab`, `dd2cdde`, `7e7fd70873`, and `d268cba9f5` by removing unnecessary data about each tally participation from staking positions and increases the maximum size of a staking UTXO
- **M-002**: Fixed in `23f94af` by switching to the correct algorithm
- **M-003**: Fixed in `f711886` by improving the logic for checking preservation of value in a staking state
- **M-004**: Fixed in `989afee` by adding a `winning_threshold` parameter to all tallies and a corresponding `min_winning_threshold` to the governance parameters
- **L-001**: Fixed in `75c8985` by checking that no contracts may be used in a transaction other than the treasury and value store

- **L-003**: Fixed in `60168db` by allocating 16 bytes to the tally ID for licenses
- **L-004**: Fixed in `12fe9da` by limiting UTXOs in the value store to at most 5 tokens

The uncompressed validator scripts have the following addresses:

Script name	Address
Gov state	addr1wx95yfp8vkq6jl0a77dx772v75k0w7j5av4prrmg83dn77cerxrwx
Staking	addr1w93kqcd757p2luase20z0qar8j5wgtp8u36t5stnuapf5fq6tmpcf
Tally	addr1w8mvlp5yvvxqst2ftrvnus3knvkww4hweezwjjltnruewhsw8qecj
Treasurer	addr1w8dwqhg4t85erd256wk37xj46n2qzu5d3c2eynwty0augjc3w0527
Value store	addr1wygcka3jeam0spwxfnruyt4x3wh5ml2yn8s5h8wtg3fg9acr9q0k6

The uncompressed minting policy scripts have the following addresses:

Script name	Address
Gov state NFT	5f8adf640a3f69f860b5d34006c34eba1144db2040e3e0f4b77a246a
Staking vote NFT	dab1013c88547bbdb2cb8bc65488d11176a4c916c242a0b91846b77f
Vault FT	b8300b7d34d4920adeac6e38c367e32aec5908bf6ba56d2ae7d2eec0
Vote permission NFT	8873a926c999e1c9f6ed1b07664f58e183ca56a64c5c16b234dacc64
Tally auth NFT	2b13b4def00bbea17a3a4f113e9f46335e0cfa0ab76cd3aec9c31d99

Script name	Address
Treasurer NFT	c42c5f9e14c712b954b6b78c9e775e6c5416ec19a8bb68cb0e0c3345

The compressed validator scripts have the following addresses:

Script name	Address
Gov state	addr1w9mnw8ngvvyezngv8ha56m6ryny5s0nfcuvqkwld0gq6j3g8fa4ax
Staking	addr1w9klu6cz9y0d0f3u9e4travkzvzfsl8gv8ay8dpwmd6pr9ssa8jwk
Tally	addr1wy83n33mrw200szwy26dpvcycfu8gr05ajyk0h7znncsvfcfu9nda
Treasurer	addr1w9pjhu7vd7n3lqjx3rdegqt87s9q3lm2pm367k9qyuy3xugy0jcuu
Value store	addr1wxexf93pzh55w3eyw3h4sa5t3tut3w6848txe00fpeefhhc3a7shl

The compressed minting policy scripts have the following addresses:

Script name	Address
Gov state NFT	addr1w8p9s9afhynw7jkd46z52ypueurswzrthylu07uux5q2f9gulmvg3
Staking vote NFT	addr1w9fu82q8gxanxpys736ezdmzy76jmywv6pnlxqavtrfcrngm2zdrj
Vault FT	addr1w9xzzzth2kefenmgp7k3r8jy7dkauyyexerlt2n72lwppacpta79p
Vote permission NFT	addr1w9zpj0ssylv9xg3vz52xf7gznm0akq33s6uyw60alh79hdcnktz73
Tally auth NFT	addr1w87ntg2l74s2c9cjg93un24fu0tw2nhuhl89wj0gz7n027gy750wc

Script name	Address
Treasurer NFT	addr1w8p9s9afhynw7jkd46z52ypueurswzrthylu07uux5q2f9gulmvg3