macro

PartyDAO A-2

Security Audit

Feb 7, 2022
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for PartyDAO's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from January 16, 2023 to January 27, 2023.

The purpose of this audit is to review the source code of certain PartyDAO Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	1	-	-	1
Medium	1	-	-	1
Low	1	-	-	1
Code Quality	4	-	2	2
Informational	1	1	-	-

PartyDAO was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Telegram with the PartyDAO team.
- Available documentation in the pull request.

Source Code

The following source code was reviewed during the audit:

- **Pull Request:** Version 1.1
- **Commit Hash:** `991cf2f4bb9ff785a39a2fb4dd19900beaa2cfbc`
- **Details:**
  These are new features and modifications implemented in the v1.1.
  - The new crowdfund type that can buy multiple NFTs and enter the governance stage with them (CollectionBatchBuyCrowdfund)
  - The new crowdfund that can continue bidding on another auction of the same collection if it doesn't win instead of ending (RollingAuctionCrowdfund)
  - Addition of contributeFor() to crowdfunds to allow contributing on other's behalf
  - Addition of ability for crowdfunds to set min/max contributions per address
  - Allowing host to bid custom amounts in case of auction crowdfunds
  - Support for ERC1155 listing on OpenSea
  - Support for dutch auction listings on OpenSea
  - Removal of useless listPrice param from Fractional proposals
- **Inside ./contracts**

Contract	SHA256

./crowdfund/AuctionCrowdfundBase.sol	`41028f6f6c1ddf7881dc318cda77d194b38ce56d05d967d30d58987351991a9c`
./crowdfund/AuctionCrowdfund.sol	`911e0d5f062750cb3966a50edc1e330fd2c79d1405ee07086ff55ed8e0399e8f`
./crowdfund/BuyCrowdfundBase.sol	`608343c4164455f252aa844c308bec75f7b8e433398038cde479fdf70342f9d3`
./crowdfund/BuyCrowdfund.sol	`75a6374b214f225011cd32806efba8bde60f84472cea714a23152f14883e89ec`
./crowdfund/CollectionBatchBuyCrowdfund.sol	`b255ad7e844e8a0d40afeb24bafea3ccebcd4a048026290fc55cf0a004a9199b`
./crowdfund/CollectionBuyCrowdfund.sol	`e10e6f91e4618f1ebf569d0b6754c03f1c073af4e5c54237a7197f408e0329eb`
./crowdfund/CrowdfundFactory.sol	`8ca6c193a066205309ba4441393972abd21061bb583bfee3d99891e6b4382e2d`
./crowdfund/Crowdfund.sol	`97d090dec67daafdc07fde0f9c9461335eaa90a076aceb37d1525f18e4a23e7e`
./crowdfund/RollingAuctionCrowdfund.sol	`4a217ead4206766fb19fe37591f5714c8dc7456a24e4e38e40d7d820b06e6275`
./globals/LibGlobals.sol	`44b58fa6efba00a48c7832d035eecc09b649c598118754597d2b137295bebcdb`
./proposals/FractionalizeProposal.sol	`1d629273470328a0475ee3b24094e3cacac705e33af8a86a0572e731d9060c61`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ `CollectionBatchBuyCrowdfund`: Hosts can use contributions without distributing voting power

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec-Breaking	Fixed	High	Medium

`CollectionBatchBuyCrowdfund` contracts allow users to buy multiple NFTs and enter the respective governance phase with them. To do so, hosts need to call `batchBuy()` and buy at least one NFT successfully.

Since the `batchBuy()` function logic allows the execution of multiple `_buy()` attempts without reverting (even if the NFT wasn't acquired), hosts can execute arbitrary calls using any value and use the crowdfunding balance as they wish, with the only condition that at least one collection NFT is acquired.

As we can see inside `batchBuy()` logic in line 153, if the call executed didn't buy the respective NFT id (`args.tokenIds[i]`), the value used (`args.callValue[i]`) won't be accounted for in the logic, and the `for` loop will continue.

```
for (uint256 i; i < args.tokenIds.length; ++i) {
    if (root != bytes32(0)) {
        // Verify the token ID is in the merkle tree.
                ...
    }

    // Execute the call to buy the NFT.
    (bool success, bytes memory revertData) = _buy(
        token,
```

```
            args.tokenIds[i],
            args.callTargets[i],
            args.callValues[i],
            args.callDatas[i]
        );

        if (!success) {
            if (args.minTokensBought >= args.tokenIds.length) {
                // If the call failed with revert data, revert with that data.
                if (revertData.length > 0) {
                    revertData.rawRevert();
                } else {
                    revert FailedToBuyNFTError(token, args.tokenIds[i]);
                }
            } else {
                // If the call failed, skip this NFT.
                continue;
            }
        }

        totalEthUsed += args.callValues[i];

        ++tokensBought;
        tokens[tokensBought - 1] = token;
        args.tokenIds[tokensBought - 1] = args.tokenIds[i];
    }
```

This allows hosts to use the crowdfunding balance without distributing the equivalent voting power. Contributors would not be able to claim their respective refunds nor have voting power in the formed party.

Remediation to Consider:

Consider checking whether failed `_buy()` calls changed the balance of the contract and revert if so, disallowing using ETH if no NFT was acquired.

~~M-1~~ `RollingAuctionCrowdfund`: While moving to a new auction, the maximum bid value is not being updated/ checked

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	Medium	Medium

`RollingAuctionCrowdfund` allows contributors to roll into the next auction for new NFT if the current auction is not successful, without starting the new crowdfunding altogether. The goal is to be able to bid until you acquire NFT. However, there is no option for them to update the already set maximum bid in initialization.

There could be a case where the next auctioned NFT is either worth less or more as per the consensus of contributors than the initially set maximum bid. There could also be the case when the minimum bid allowed for the next auction currently is more than the already defined maximum bid since the check of a minimum bid is only done in initialize.

Remediation to Consider:

- Consider making maximum bid auction specific. This could be done by passing one more parameter as `nextMaximumBid` inside `finalizeOrRollOver().`
- As a preventive measure, check whether the next auction's selected minimum bid is lower than the maximum bid, as is currently being done in the crowdfunding initialization. If the minimum required bid for the auction is higher than the maximum bid, revert.

~~L-1~~ Auction Crowdfunds can reach an impasse: Contributors would need to wait for the auction to conclude even when they cannot acquire the NFT

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	Low	Low

When calling `finalize()`, expired `AuctionCrowdfunds` and `RollingAuctionCrowdfunds` contracts will attempt to finalize the auction if a bid was previously placed (`lastBid != 0`). If the crowdfund is not the highest bidder, contributors would need to wait for the auction on external markets to conclude even when crowdfund is expired and there is no possibility for them to acquire the NFT.

To be more detailed, in this scenario, the crowdfunding:

- Won't be able to place a new bid on the auction.
- No further contributions can be added.
- Cannot be finalized until the auction duration expires.

Remediations to Consider:

Consider allowing auction crowdfunds to get concluded when they have expired and are not the current highest bidder (`lastBid != 0`)

~~Q-1~~ `CollectionBatchBuyCrowdfund`: Lack of check on the length of `BatchBuyArgs` parameters inside `batchBuy()`

TOPIC	STATUS	QUALITY IMPACT
Input Validation	Fixed	Medium

`tokenIds`, `callTargets`, `callDatas`, `callValues`, `Proofs` all arrays need to be of

the same length. Consider either adding explicit checks or combining these in a separate struct.

```
struct BatchBuyArgs {
        uint256[] tokenIds;
        address payable[] callTargets;
        uint96[] callValues;
        bytes[] callDatas;
        bytes32[][] proofs;
        uint256 minTokensBought;
        uint256 minTotalEthUsed;
        FixedGovernanceOpts governanceOpts;
        uint256 hostIndex;
}

struct xxxxx
{
        uint256 tokenIds;
        address payable callTargets;
        uint96 callValues;
        bytes callDatas;
        bytes32[] proofs;
}

struct BatchBuyArgs
{
        xxxxx[] ____
        uint256 minTokensBought;
        uint256 minTotalEthUsed;
        FixedGovernanceOpts governanceOpts;
        uint256 hostIndex;
}
```

Q-2 Redundant tracking of tokens array inside `CollectionBatchBuyCrowdfund`

TOPIC	STATUS	QUALITY IMPACT
Redundant State Updates	Wont Do	Medium

Due to the nature of the collection batch crowdfund, it is given that the NFT being bought is always going to be of the same collection(address) but with different token ids; hence there is no need to populate the tokens array with the same token address multiple times and using this array as a measure of NFTs owned everywhere else.

```
function batchBuy(BatchBuyArgs memory args) external onlyDelegateCall retu
      ________
      IERC721[] memory tokens = new IERC721[](args.tokenIds.length);
      IERC721 token = nftContract;
      bytes32 root = nftTokenIdsMerkleRoo;
      for (uint256 i; i < args.tokenIds.length; ++i) {
           ______
            tokens[tokensBought - 1] = token;
            args.tokenIds[tokensBought - 1] = args.tokenIds[i];
            emit console(args.tokenIds);
      }
      __________

      assembly {
            // Update length of `tokens`
            mstore(tokens, tokensBought)
            // Update length of `tokenIds`
            mstore(args.tokenIds, tokensBought)
      }
      ____________
      return
            _finalize(
                tokens,
                args.tokenIds,
                totalEthUsed,
                args.governanceOpts,
                // If `_assertIsHost()` succeeded, the governance opts wer
                true
      );
    }
```

RESPONSE BY PARTYDAO

> There will be other crowdfund types built in the future that do not acquire NFTs all from the same collection that require Party initialization to accept an array of tokens.

Q-3 Lack of check on the custom bid

TOPIC	STATUS	QUALITY IMPACT
Input Validation	Wont Do	Low

From Party Protocol version 1.1, custom bids are enabled for auction crowdfund types, but consider reverting if the passed bid is less than the minimum bid auction expects to avoid unnecessary computation.

RESPONSE BY PARTYDAO

> Rather not add the gas overhead to add a check for something that is already handled by the market contracts.

~~Q-4~~ Nitpicks

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

1. Incorrect Comment

```
Crowdfund.sol
L197
// If this contract has ETH, either passed in during deployment or
// pre-existing, credit it to the initialContributor.
Only `msg.value` is being used as initialContribution.

L349
/// @notice contributeFor() in batch form.
/// Will not revert if any individual burn fails.

L354
/// @param revertOnFailure If true, revert if any burn fails.
```

1. Duplicate Comment

```
RollingAuctionCrowdfund.sol
L196

if (nftContract_.safeOwnerOf(nftTokenId_) == address(this) && lastBid_ !=
// Create a governance party around the NFT.
party_ = _createParty(governanceOpts, false, nftContract, nftTokenId);
// Create a governance party around the NFT.
```

3. Unused Library Imported

```
CollectionBatchBuyCrowdfund.sol
L5
import "solmate/utils/MerkleProofLib.sol";
```

1. Test missing modifier

```
FractionalizeProposalForkedTest.t.sol
L173
```

```
Should have the `onlyForked` modifier
function testForked_canRedeem() public {
```

I-1 Additional Analysis

TOPIC	STATUS	IMPACT
Informational	Acknowledged	Informational ✳

These are not exploitable issues **but are important constraints/limitations/leads to consider.** They might not be exploitable currently, but that may not remain the case if any component behavior changes in the future. Our main purpose here is to make you more informed about any future choices you make.

`CollectionBatchCrowdfund`:

1. `TotalEthUsed` in call for buy must be > 0 for it to succeed. Hence the occurrence of a gift case of `BuyCrowdfundBase` would not be possible. This sanity check saves from exposure on various edge cases with gifting.
2. One can pass the same `tokenIds` again and inflate tokens bought to pass the condition of minimum token bought without doing anything in subsequent calls; however, creating a party would revert since you can't transfer the same NFT multiple times from CF to party.
3. One can transfer the NFT out in the next call of `buy` after satisfying the constraint of ownership in the first call; however, create party would revert since now there would be no NFT for transfer to the party.

`ListOnOpenseaProposal`:

1. If the conduit is updated in between, the `ERC1155` approval will be cleared for the updated conduit but will still remain active for the old one. However, one won't be able to exploit this since the call to `validate` is blocked in arbitrary proposals.

2. `setApprovalAll` done for `ERC1155` is not cleared if the proposal turns successful, leaving the approval in place if the party owns other `ERC1155` tokens of the same collection. However, this also one would not be able to exploit since the call to `validate` is blocked in arbitrary proposals.

3. Wrapping a precious token with `tokenType.ERC1155` in an `OpenSeaProposal` will skip the precious token checks, calling `setApprovalForAll`, and `validate` in an invalid order without unanimous votes. The placed order would not be fillable since the token type won't match the actual type, and will fail when trying to fill the order.

4. By using `TokenType.ERC1155` as token type for precious token one can bypass the Zora check and directly list on the open sea, since `setApprovalAll` is present in both `ERC721`, and `ERC1155` with the same signature; however, one won't be able to exploit it since the seaport contract would revert in transfer module due to difference in signatures for `safeTransferFrom` for `ERC721` and `ERC1155`.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.