Fragmint

Final Audit Report

April 1, 2022

Team Omega

teamomega.eth.limo

Summary

Evedo has asked Team Omega to audit the contracts that define the behavior of the Fragmint contracts.

On March 18, we delivered a preliminary report. We found **4 high severity issues** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **3** issues as “medium” - these are issues we believe you should definitely address. In addition, **7** issues were classified as “low”, and **12** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Evedo subsequently addressed the issues, and we reviewed the fixes in the current report. All issues were resolved.

Severity	**Number of issues**	**Number of resolved issues**
High	4	4
Medium	3	3
Low	7	7
Info	12	12

Scope of the Audit

Code from the following three repositories was audited:

- `https://github.com/evedo-co/frag-tge/commit/bbd61cf2147aea4cd8de2fc6cebedfd130bcf02d`
- `https://github.com/evedo-co/fragmint-auction-sc/commit/dc54cc7ade03ded5804491a9eb41486b2dba96e0`
- `https://github.com/evedo-co/fragmint-nft-contract/commit/1fc9728c5d0762f7b543115558edc00301bf4906`

And specifically the following Solidity contracts:

```
fragmint-auction-sc/contracts/FragAuction.sol
fragmint-nft-contract/contracts/NFT.sol
fragmint-nft-contract/contracts/NFTFactory.sol
frag-tge/contracts/Fragmint.sol
```

Resolution

After processing our report, Evedo addressed the issues in the following commits. We have reviewed the fixes.

- `https://github.com/evedo-co/frag-tge/commit/a608075ed7edbf73b62c5aab0e618c87036a313e`
- `https://github.com/evedo-co/fragmint-auction-sc/commit/a608075ed7edbf73b62c5aab0e618c87036a313e`
- `https://github.com/evedo-co/fragmint-nft-contract/commit/a608075ed7edbf73b62c5aab0e618c87036a313e`

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues, concerning mostly the solidity version setting and functions visibility, were found and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Configure continuous integration [info] [resolved]

In none of the three repositories, continuous integration configured - i.e. test and code quality scripts are not run automatically on each deployment.
Severity: Info
Recommendation: We strongly recommend configuring automatic tests and coverage reports using github, for example.
Resolution: This issue was resolved as recommended.

G2. Test coverage is incomplete [low] [resolved]

Across the inspected repositories, test coverage is very incomplete. In addition, the coverage is not properly configured to ignore test and mock files, distorting the result of the coverage. In the `fragmint-auction-sc` repository, part of the tests are ran against a separate contract, `fragAuctionMock.sol`, which is an almost exact copy of `fragAuction.sol`.

The coverage results for the relevant contracts are:

```
-------------------|----------|----------|----------|----------|----------------|
File               |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------|----------|----------|----------|----------|----------------|
 contracts/        |     9.09 |     5.56 |    10.71 |     8.97 |                |
  Fragmint.sol     |       50 |       50 |       50 |       50 |... 51,52,53,56 |
  TokenVesting.sol |        0 |        0 |        0 |        0 |... 338,339,348 |
-------------------|----------|----------|----------|----------|----------------|
All files          |     9.09 |     5.56 |    10.71 |     8.97 |                |
-------------------|----------|----------|----------|----------|----------------|
```

```
-----------------|----------|----------|----------|----------|----------------|
File             |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------|----------|----------|----------|----------|----------------|
 contracts/      |    96.43 |       75 |    94.44 |    96.55 |                |
  NFT.sol        |       95 |      100 |    92.86 |       95 |             95 |
  NFTFactory.sol |      100 |       50 |      100 |      100 |                |
-----------------|----------|----------|----------|----------|----------------|
All files        |    96.43 |       75 |    94.44 |    96.55 |                |
-----------------|----------|----------|----------|----------|----------------|
---------------------------|---------|----------|---------|----------|----------------|
File                       | % Stmts | % Branch | % Funcs |  % Lines |Uncovered Lines|
---------------------------|---------|----------|---------|----------|----------------|
 contracts/                |   51.95 |    41.67 |   57.14 |    51.95 |                |
  BpLibrary.sol            |     100 |      100 |     100 |      100 |                |
  FragAuction.sol          |   51.32 |    41.67 |   53.85 |    51.32 |...192,194,195 |
  IFragAuction.sol         |     100 |      100 |     100 |      100 |                |
 contracts/mockedContracts/|   88.16 |    61.76 |   88.24 |    88.16 |                |
  FragAuctionMock.sol      |   89.04 |    61.76 |   92.86 |    89.04 |...120,122,152 |
  NFT.sol                  |   66.67 |      100 |   66.67 |    66.67 |             25 |
---------------------------|---------|----------|---------|----------|----------------|
All files                  |   69.93 |    51.43 |   74.19 |    69.93 |                |
---------------------------|---------|----------|---------|----------|----------------|
```

Severity: Low
Recommendation: Properly configure the coverage to ignore tests and mock files. Extend tests to cover all of the codebase.
Resolution: This issue was resolved. Test coverage is now properly configured and has reached 100% across all repositories.

G3. Pin versions of solidity dependencies [low] [resolved]

In all three repositories, `package.json` specifies a range of versions of the OpenZeppelin dependencies rather than a single one. This can lead to unexpected problems, when other developers, or yourself at a later date, will recompile the contracts, as it is not explicitly clear from `package.json` which version of the dependencies is used for compiling the contracts.

For example, in fragmin-auction-sc, the version specified in `package.json`

```
      "@openzeppelin/contracts": "^4.3.2"
```

According to `package-lock.json` the version used for the latest build of the repository is 4.4.0. This is a deprecated version and should be upgraded.
Recommendation: Specify fixed versions of smart contract dependencies in package.json instead of ranges, so that the solidity code can be verified and there is no ambiguity about the actual code you are to deploy or already have deployed.
Resolution: The issue was resolved as recommended.

G4. Licensing information is incomplete [low] [resolved]

The contracts under review all contain an SPDX-License identifier identifying the code to be released under the MIT license:

```
// SPDX-License-Identifier: MIT
```

However, a copy of the MIT license is not included in the repositories. The MIT license is not just a descriptive text: it is a text in which the author claims the copyright and then grants rights to users of the software, and therefore its inclusion is essential for the license to have its full legal effect.
Severity: Low
Recommendation: Include a copy of the MIT license in each of the repositories.
Resolution: This issue was resolved as recommended.

G5. Update README.md files with proper information [info] [resolved]

In 2 of the 3 repositories, the `README.md` file contains just the auto-generated hardhat text, and the 3rd one (of the auction) is short and contains wrong instructions. These should be updated with up-to-date instructions and documentations.
Severity: Info
Recommendation: Update the `README.md` files with relevant and up-to-date documentation and instructions.
Resolution: This issue was resolved. The `README.md` files now contain up-to-date documentation for each of the repositories.

FragAuction.sol

A1. NFT that did not get any bids will be stuck in the contract [high] [resolved]

The auction contract has only a single way to send out an NFT, and that is to the winner of the auction (line 108). However, if no bids for the NFT were sent to the contract, the original depositor has no way to claim back their NFT, which will be stuck forever in the auction contract.
Severity: High
Recommendation: Allow the depositor of the NFT to withdraw the NFT if the auctions ends and no bids were placed.
Resolution: This issue was resolved. The depositor of the NFT can now claim it back after the auction has ended if no bids were made for it.

A2. ETH could get stuck in the contract if not enough shares were assigned [high] [resolved]

The `startAuction` function requires that the total shares assigned will be lower than or equal to 10,000. When the auction is finished and a winner is selected, each share gives its owner exactly 1/10000 of the winning bid, regardless of the total amount of shares in existence. If the shares assigned are lower than 10,000, then the remaining funds from the winning bid will be stuck in the contract, and not redeemable by anyone.
Severity: High
Recommendation: Change the requirement in line 50 to require the `sharesSum` will be equal to 10,000, or, alternatively, store the value of `sharesSum` and give each user a part of the winning bid proportional to their share.
Resolution: This issue was resolved. The `require` statement now checks 100% of shares were issued.

A3. Last bidder's ETH and the auctioned NFT could get stuck if a shareholder is malicious [high] [resolved]

On selecting a winner by calling `selectWinner`, the function `transferRevenue` is called. This function loops over all shareholders, and will distribute the ether of the winning bid to each of the shareholders by calling `shares[i].account.transfer(feeValue)`
If one of these transfers fails, then the whole transaction fails. A malicious shareholder could abuse this fact by creating a contract at the address of `shares[i].account` that fails when Ether is sent to it. If this is the case, both the NFT as well as the ether sent with the highest bid will be stuck in the contract. This does not need to be a griefing attack: an attacker could use this to extort a sum up to twice the value of the NFT with a contract such as the following:

```
contract Attack {
    address payable attackerAddress;
    uint256 ransomValue;

    constructor(address payable _attackerAddress, uint256 _ransomValue) {
        attackerAddress = _attackerAddress;
        ransomValue = _ransomValue;
    }

    receive() external payable {
        assert(false);
    }

    function deleteContract() public payable {
        require(msg.value == ransomValue);
        selfdestruct(attackerAddress);
    }
}
```

Note that the attacker could provide the address of the contract before deploying the actual contract, so there is no way for the auction creator to detect the possibility of this attack - the address would not be distinguishable from a normal user account at the time of registration of the auction.
Severity: High
Recommendation: Separate the `transferRevenue` logic from winner selection, and allow individual shareholders to redeem their shares individually (or in group/ range specified by the redeemer). See also A6.
Resolution: This issue was resolved. Winner is now assigned to the highest bid automatically and separately from revenue distribution, also, the individual shareholders can now claim their revenue independently of the other shareholders.

A4. Winner of auction could auction NFT that is not deposited in the contract [high] [resolved]

In the `claimReward` function, the `safeTransferFrom` call to transfer the NFT to the winner is being done before the `rewardClaimed` flag is set to true. The function `safeTransferFrom` calls `onERC721Received` on the receiver of the NFT, if the receiver is a contract, as a way of checking that the receiver is capable of handling NFTs.
This allows a reentrancy attack where the winner can create a new auction for this NFT without transferring the NFT to the auction contract. The attack works as follows:

- The attacker receives the NFT into a contract that has a malicious `onERC721Received` function
- The malicious function calls the `startAuction` function to re-auction the NFT. The NFT is now transferred back to the Auction contract
- The malicious function calls the `claimReward` function with the auction ID of the previous auction (the original one where the winner won the NFT). Because the `rewardClaimed` flag will still be false (it is updated only after the token is transferred to the winner), this call is successful, and the NFT is transferred back from the contract to the attacker

So the result is that the winner has now created a new auction for the NFT, but the NFT is not owned by the auction contract. This allows the attacker to trick bidders into bidding on an NFT they could not claim.
Severity: High
Recommendation: Perform state changes, in this case, update the `rewardClaimed` flag, before making external calls.
Resolution: This issue was resolved as recommended.

A5. All highest bids and NFTs could be stuck in the contract if the owner is malicious or unresponsive [medium] [resolved]

In case the key of the owner address of the contract is compromised or lost for any reason (a hack for where it's stored, failure of systems, etc.), the `selectWinner` function will be impossible to call. This would mean that auctions cannot be ended because winners cannot be selected (as the `selectWinner` function is only callable by the owner of the contract) and so all the NFTs in the contract, and each highest bid of each auction, would be stuck forever. This could allow ransom attack in case the key is compromised, where the hacker refuses to call the `selectWinner` function until they get a ransom, or it could cause a loss of the NFTs and the deposited highest bids in case access to the key was lost.
Severity: Medium
Recommendation: Allow canceling an auction in case a winner has not been selected for a predetermined time (for example 30 days) after the end time of the auction was reached.
Resolution: This issue was resolved. The highest bidder is now automatically assigned as the winner, so auctions are now automatically finalized once their end time passes.

A6. Last bidder's ETH, and the NFT could get stuck in the contract if too many shareholders were assigned [medium] [resolved]

In the `startAuction` function, there's an effective limit of 10,000 entries in the `sharesPerAuction` array, however, if too many shareholders are registered, the `selectWinner` function will run out of gas sending the auction winning bid to too many addresses. This means selecting a winner would be impossible, and so both the NFT and the ETH of the last bidder will be stuck in the contract.
Note that even if the transaction does not run out of gas, but simply consumes a lot of it, the cost of transferring the rewards to each of the shareholders could become very high and disproportional to the value of the auction.
Severity: Medium - although loss of funds is possible, the scenario of having so many shareholders seems quite improbable.
Recommendation: Separate the `transferRevenue` logic from winner selection, and allow individual shareholders to redeem their shares individually (or in group/ range specified by the redeemer). See also A3.
Resolution: A limit of up to 10 shareholders per auction was added, also, shareholders can now claim their revenue independently of other shareholders.

A7. Missing sanity checks on auction start and end time [low] [resolved]

In the `startAuction` function, there are no sanity checks that the start time is lower than the end time, nor that the end time is lower than the current time, or that the start and end time are not too distant into the future. These sanity checks are important to prevent errors and unexpected behavior.
Severity: Low
Recommendation: Implement sanity checks requiring that the end time is in the future and greater than the start time of the auction. Also make sure that the auction start and end time are not set too far into the future.
Resolution: This issue was resolved. Sanity checks were added to ensure the start time is neither in the past nor in more than 31 days, and that the end time is at least 1 hour after than the start time but no more than 31 days after it.

A8. NFT could get stuck if bid is sent from a contract not supporting ERC721Receiver standard [low] [resolved]

If the winning bidder is a contract that does not implement the `ERC721Receiver` interface (cf. https://eips.ethereum.org/EIPS/eip-721), then attempts to transfer a ERC721 token to such an address will fail. This means that a user that uses a contract to participate in the auction could win the bid but not be able to receive the NFT, and the NFT will be irrecoverably stuck in the auction contract.
Severity: Low - although the issue can lead to loss of funds, it is quite unlikely that a contract with exactly these properties would be used for bidding.
Recommendation: One way of resolving the issue is to add an optional beneficiary argument to the `claimReward` function, so that the winner of the auction can decide where to send the NFT.
Resolution: This issue was resolved as recommended.

A9. transferRevenue function is overly complex [low] [resolved]

The `transferRevenue` function calls multiple internal functions only to make a simple calculation, it could be simplified for improved readability and gas saving.

The implementation looks like this:

```
function transferRevenue(uint256 totalAmount, Shares[] memory shares)
internal {
        uint256 restValue = totalAmount;

        for (uint256 i = 0; i < shares.length; i++) {
            (uint256 newRestValue, uint256 feeValue) =
subSharesInBp(restValue, totalAmount, shares[i].value);
            restValue = newRestValue;
            if (feeValue > 0) {
                shares[i].account.transfer(feeValue);
            }
        }
    }
```

This implementation can be simplified considerably by refactoring it as follows (which will allow for considerable simplification of the code to remove the `subSharesInBp` function and the `BpLibrary.sol` dependency)

```
function transferRevenue(uint256 totalAmount, Shares[] memory shares)
internal {
        for (uint256 i = 0; i < shares.length; i++) {
            uint256 feeValue = totalAmount * shares[i].value / 10000
            if (feeValue > 0) {
                shares[i].account.transfer(feeValue);
            }
        }
    }
```

Recommendation: Refactor the function as described to gain in clarity, save some gas, and reduce the risk of errors.
Resolution: This issue was resolved. The calculation of shareholders' revenue is now done as suggested above.

A10. Follow Check Effects Interaction Pattern [info] [resolved]

In multiple places across the code, there are calls to external addresses being done before making state changes. It is best practice, and more future proof for changes that may be done, to first make state changes and only then do external calls. The relevant parts in the code are:

`startAuction` - external call at line 30 then state changes at lines 31 to 49 (we do not see an attack vector here)
`claimReward` - external call at line 108 then state changes at line 109 (see issue A4 for a description of a possible attack)
Severity: Info
Recommendation: Perform state changes before making external calls.
Resolution: This issue was resolved as recommended.

A11. Unnecessary require on auction existence [info] [resolved]

In the `placeBid` function, there are 2 checks, the first that the auction exists, and the second that it is active. This first check of the auction existence is completely redundant, as the second check will necessarily fail if the auction does not exist.
Severity: Info
Recommendation: Remove the require at line 56 and delete the now unused `checkAuctionExistence` function.
Resolution: This issue was resolved as recommended.

A12. Avoid loading to memory data that is read only once [info] [resolved]

Both in line 58 (the `_bid` variable) and in line 116 (the `auction` variable) are only read once, and instead of being loaded to memory, their value could be explicitly written where they are used,
Severity: Low - this will save some gas
Recommendation: Remove the above mentioned variables and instead just use their values where they are needed.
Resolution: This issue was resolved. Variables that are read only once are no longer being loaded to memory.

A13. Load variable to memory instead of storage when possible [info] [resolved]

On line 148 the `currentAuction` variable is being loaded to storage but never changed, which means it could just be loaded to memory instead.
Severity: Low - some gas can be saved
Recommendation: Load the `currentAuction` variable to memory instead of storage.
Resolution: This issue was resolved as recommended.

Fragmint.sol

F1. transferFrom to the burning address will try to burn tokens from the sender instead of the token holder [medium] [resolved]

If transferFrom is called with the `BURNING_ADDRESS` as the to parameter, the function will try to spend the tokens of the msg.sender, rather than the tokens of the from address. This is unexpected and could lead to a user accidentally burning her own tokens.
Severity: Medium. This breaks ERC20 compatibility. A user could call this function by mistake and burn her own tokens. A third-party contract that counts on `transferFrom` transferring the tokens from the `from` address rather than her own could be tricked into burning her own tokens in a griefing attack.
Recommendation: Consider removing the "burn tokens by sending them to 0xDEAD" logic altogether.
Resolution: This issue was resolved. The special burning logic has been removed altogether.

F2. Transfer to 0xDead will fire Transfer to 0x0 event [low] [resolved]

Calling `transfer(to, amount)` with to == 0x000000000000000000000000000000000000dead will directly call the `_burn` function from OpenZeppelin's ERC20 implementation instead of calling OpenZeppelin's implementation of `transfer`. Burning tokens in the OpenZeppelin implementation, burning tokens fires an event of the form `Transfer(msg.sender, address(0), amount)` - i.e. a transfer to the zero address - as a way of signaling that the amount of tokens were burnt.
Although both of these effects of calling `transfer` for this specific address are still compatible with the ERC20 standard (we believe), they are unexpected and in theory clients may expect a different behavior.
Severity: Low
*Recommendation:*Consider removing the "burn tokens by sending them to 0xDEAD" logic altogether.
Resolution: This issue was resolved. The special burning logic has been removed altogether.

F3. Make fragmintDistributionAddress immutable [info] [resolved]

The value of `fragmintDistributionAddress` is set in the constructor and can never be changed thereafter, and can therefore be declared `immutable`.
Severity: Low
Recommendation: Mark the `fragmintDistributionAddress` variable as `immutable.`
Resolution: This issue was resolved as recommended.

F4. Make pause and unpause functions external [info] [resolved]

The `pause` and `unpause` functions are not called within the contract itself and could be declared `external`.
Severity: Info
Recommendation: Declare the `pause` and `unpause` functions as `external` instead of `public`.

NFT.sol

N1. transferOwnership function can be removed [info] [resolved]

The `transferOwnership` function does nothing but call `super.transferOwnership`, and so can be safely removed without altering the functionality.
Severity: Info
Recommendation: Remove the unnecessary `transferOwnership` function.
Resolution: This issue was resolved as recommended.

NFTFactory.sol

NF1. Use OpenZeppelings clone() function instead of copying the code [info] [resolved]

The `_clone()` function in NFTFactory.sol is an exact copy of the `clone()` function in the `@openzeppelin/contracts/proxy/Clones.sol` library, which could be used instead to avoid code duplications and benefit from future improvements or fixes in the OZ code.
Severity: Info
Recommendation: We recommend not copying code that is already available in the repository, but using the library instead.
Resolution: This issue was resolved as recommended.

NF2. Make implementation upgradeable or immutable [info] [resolved]

The `implementation` variable has no setter after being initialized in the constructor. We believe it would be useful to allow the factory owner to update the value of the `implementation`, or if you prefer it immutable, it could be marked `immutable` to save gas.
Severity: Info

Recommendation: Make the `implementation` variable updateable by the owner of the factory, or mark it `immutable`.
Resolution: This issue was resolved. The `implementation` variable is now properly marked as `immutable`.

NF3. Make getClones function external [info] [resolved]

The `getClones` function is not called within the contract itself and could be declared `external`.
Severity: Info
Recommendation: Declare the `getClones` function as `external` instead of `public`.
Resolution: This issue was resolved as recommended.