yAcademy yFu NFT review

Review Resources:
None beyond the code repositories

Residents:

- blockdev
- NibblerExpress

Table of Contents

Review Summary

yFu NFT

yFu is an ERC721 NFT collection where each token represents the same metadata. Minters will be able to claim a physical comic book.

The main branch of the yfu-contracts Repo was reviewed over 6 days. The contract was reviewed from September 1 to September 6, 2022. The code was reviewed by 2 residents for a total of 17 hours (blockdev 13 hours, NibblerExpress 4 hours). The review was limited to one specific commit.

Scope

Code Repo,
Commit

The commit reviewed was edacf43d777a93711530b81a829099bb885f498e. The review covered the entire repository at this specific commit but focused on the only contract in the repository.

After the findings were presented to the yFu team, fixes were made and included in several PRs.

The review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAcademy and the residents make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAcademy and the residents do not represent nor imply to third party users that the code has been audited nor that the code is free from defects. By deploying or using the code, yFu and users of the contracts agree to use the code at their own risk.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements,
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Medium Findings

1. Medium - No function to freeze transfers after unfreezing them (blockdev)

NFT transfers are paused when `transfers_frozen = true`. Such is the case on contract deployment. Addresses with `DEFAULT_ADMIN_ROLE` can turn it off through `unfreeze_transfers()` function. However, once this is called, there is no way to set `transfers_frozen` to true again. So the transfers cannot be frozen again.

Technical Details

YFUtechne.sol#L16, YFUtechne.sol#L33-L35

Impact

Medium. Once transfers are allowed, they cannot be turned off.

Recommendation

Replace `unfreeze_transfers()` with:

```
function set_transfers_frozen(bool b) public onlyRole(DEFAULT_ADMIN_ROLE) {
    transfers_frozen = b;
}
```

Developer response

This specification is intended by the product requirements. We wanted the contract to explicitly not allow transfers to be frozen again once unfrozen, so secondary market price action can flow freely without people thinking we can freeze again.

Low Findings

1. Low - Fix `supportsInterface()` (blockdev)

`YFUtechne.supportsInterface()` function overrides `ERC721.supportsInterface()` and `AccessControl.supportsInterface()`:

```
function supportsInterface(bytes4 interfaceId)
    public
    view
    override(ERC721, AccessControl)
    returns (bool)
{
    return super.supportsInterface(interfaceId);
}
```

Due to how solidity inherits contracts, calling `super.supportsInterface(interfaceId)` will call `AccessControl.supportsInterace()`. So for `type(IERC721).interfaceId`, this function will return `false`.

Technical Details

YFUtechne.sol#L65-L72

Impact

Low. Even though this NFT contract implements ERC721 interface, `supportsInterface()` called on ERC721's `interfaceId` will return `false`.

Recommendation

Apply this diff:

```
function supportsInterface(bytes4 interfaceId)
    public
    view
    override(ERC721, AccessControl)
    returns (bool)
{
-    return super.supportsInterface(interfaceId);
+    return ERC721.supportsInterface(interfaceId)
+        || AccessControl.supportsInterface(interfaceId);
}
```

Developer response

Fixed in PR#8

2. Low - NFT minting might be unfair if demand is high (blockdev)

The NFT mint function has no restrictions on address who can mint, and the number of tokens minted per address. If the demand of this collection is high, someone can mint all tokens to themselves in the worst case.

Technical Details

YFUtechne.sol#L41-L50

Impact

Low. The NFT team confirmed that this should not be an issue in practice. If it happens, they'll re-deploy the contract, perhaps after addressing this issue.

Recommendation

Consider adding an address allowlist for minting. This is achieved through hashing all the addresses in a merkle tree, and storing the merkle root in the contract. Then add a max limit on total mints per address in the allowlist.

Developer response

Acknowledged. We are fine with this issue.

3. Low - Add sweep functions for Ether and ERC20 tokens (blockdev)

In case, any Ether or ERC20 end up in this NFT contract's balance, (due to user error or incorrect royalty settings on marketplaces), there's no way to recover those funds.

Technical Details

YFUtechne.sol

Impact

Low. Funds sent to this contract accidentally will be lost.

Recommendation

Add functions `sweepEther()` and `sweepERC20()` which can only be called by addresses with special privileges (`DEFAULT_ADMIN_ROLE`). Call these functions to transfer any stuck funds out of the contract.

Developer response

Fixed in PR#8

4. Low - Use `call` to transfer Ether (blockdev)

`transfer()` uses a fixed gas stipend to the Ether receiver. This has a benefit that it doesn't forward enough gas to allow the receiver to do any malicious action. However, `transfer()` can break in case the gas cost of Ether transfer is changed in future, or if the recipient wants to perform some non-malicious action.

Technical Details

YFUtechne.sol#L45: The mint fee is transferred to `depositAddress` as follows:

```
depositAddress.transfer(PRICE);
```

Impact

Low. If `depositAddress` is a contract which consumes more gas than the allocated gas stipend, mints will fail.

The risk is limited as this address can be changed if that happens.

Recommendation

The team should estimate on how long the minting period might go on. It's unlikely that the gas cost of Ether transfer is changed in the short term. If the team sets the `depositAddress` such that it's able to receive Ether in the gas stipend from `transfer()`, it's fine to keep it as is.

Otherwise, transfer Ether as follows:

```
(bool success, ) = depositAddress.call{value: PRICE}("");
require (success, "ETH transfer failed");
```

This would let `depositAddress` to reenter the contract. However, there is no risk through reentrancy, and this address is controlled by the admin.

Developer response

Fixed in PR#8

Gas Savings Findings

1. Gas - Use `constant` and `internal` variables (blockdev)

For constant variables for which values are known at compile time, mark them as `constant` to save gas. Further, they can be marked `internal` since getter functions are created for `public` variables increasing the contract size.

Technical Details

YFUtechne.sol#L13-L14

Impact

Gas savings.

Recommendation

Replace the declarations of `MAX_SUPPLY` and `PRICE` with:

```
uint256 internal constant MAX_SUPPLY = 10;
uint256 internal constant PRICE = 1 ether;
```

Developer response

Fixed in PR#8

2. Gas - Mark public functions as `external` (blockdev)

For public functions which are not accessed from the contract itself, marking them as `external` saves gas when they are called.

Technical Details

YFUtechne.sol#L25-L50

Impact

Gas savings.

Recommendation

For the functions highlighted above, mark them as `external` instead of `public`.

Developer response

Fixed in PR#8

3. Gas - Use OpenZeppelin's `Ownable` instead of `AccessControl` (blockdev)

If there is only one role and it is only held by one address, using `Ownable` based access is gas-efficient than role based access control since instead of storing a nested map, a single storage variable is used.

Technical Details

YFUtechne.sol#L5

Impact

Gas savings.

Recommendation

- Inherit from OpenZeppelin's `Ownable` contract instead of `AccessControl`.
- Instead of `onlyRole(DEFAULT_ADMIN_ROLE)`, use `onlyOwnable`.

Developer response

Fixed in PR#8

4. Gas - Don't transfer mint fee to `depositAddress` on each mint (blockdev)

On each mint, 1 ether is transferred twice, first from minter to NFT contract, and then from the contract to `depositAddress`. This increases the gas cost of minting.

Instead, the NFT contract can hold this ether which can be sweeped at once from an address with `DEFAULT_ADMIN_ROLE`.

Technical Details

YFUtechne.sol#L45

Impact

Gas savings.

Recommendation

- Remove all functionality related to `depositAddress` in `safeMint()` function (declaration, setter function, ether transfer).
- Add a function `sweepEther()`:

```
function sweepEther(address payable _addr) external onlyRole(DEFAULT_ADMIN_ROLE) {
    (bool success, ) = _addr.call{value: address(this).balance}("");
    require(success, "!sweep");
}
```

Developer response

Fixed in PR#8

5. Gas - Use `uint256` for `_tokenIdCounter` (blockdev)

Using `uint256` instead of OpenZeppelin's `Counter` can save gas due to less overhead of incrementing `_tokenCounter`.

Technical Details

YFUtechne.sol#L11, YFUtechne.sol#L47-L49

Impact

Gas savings.

Recommendation

Declare `_tokenIdCounter` as `uint256`, and apply this diff:

```
- uint256 tokenId = _tokenIdCounter.current();
- _tokenIdCounter.increment();
- _safeMint(to, tokenId);
+ _safeMint(to, tokenId++);
```

Developer response

Fixed in PR#8

6. Gas - Explore solmate's ERC721 and ERC1155 implementation (blockdev)

There are different ERC721 implementations like from OpenZeppelin, solmate, ERC721A. ERC721A gives gas benefits on batch minting, but are expensive on transfers. solmate's ERC721 generally gives more gas savings than that of OpenZeppelin in every way.

Since each token has the same image metadata, also consider exploring using ERC1155.

Technical Details

solmate's ERC721.sol,
solmate's ERC1155.sol

Impact

Gas savings.

Recommendation

- Replace the use of OpenZeppelin's ERC721 implementation with solmate's ERC721 implementation. This is certain to give gas saving.
- Consider exploring ERC1155 and compare the gas costs between the two implementations.

Developer response

Won't fix.

Informational Findings

1. Informational - Make sure to add `/` at the end of IPFS base URI (blockdev)

The `tokenURI()` function appends token ID to the IPFS base URI, hence it's important to note that the URI should have `/` at the end of it to have a well-formed `tokenURI`.

Technical Details

OpenZeppelin's ERC721.sol#L97

Impact

Informational.

Recommendation

Make sure to add `/` at the end of IPFS base URI.

Developer response

Acknowledged

2. Informational - `_beforeTokenTransfer()` should be marked as `view` (blockdev)

Functions which only read storage space should be marked as `view`. Solidity compiler also warns about this at compile time.

Technical Details

YFUtechne.sol#L52

Impact

Informational.

Recommendation

Mark `_beforeTokenTransfer()` as `view`.

Developer response

Fixed in PR#8

3. Informational - Add unit tests to gain confidence (blockdev)

Unit tests helps to catch bugs early on and future-proofs a contract from bugs that are introduced later.

Technical Details

YFUtechne.sol

Impact

Informational.

Recommendation

Use any popular development environment for unit testing and local fork testing. Some things to test for would be:

- The maximum number of tokens does not exceed `MAX_LIMIT`.
- Only token owners and approved addresses can transfer NFT.
- Fee is charged on mint.
- `tokenURI()` works correctly.
- `name` and `symbol` is correct.
- Access controlled function can only be called by approved addresses.

We have separately provided a publicly available, MIT licensed NFT repository as a reference for unit tests.

After unit testing, test the project on a test network before mainnet deployment.

Developer response

Acknowledged

4. Informational - Follow Solidity style guide (NibblerExpress)

Some function and variable names do not follow the naming conventions of the Solidity style guide (e.g., `set_deposit_address()` rather than `setDepositAddress()`). The naming conventions of the style guide are available here: https://docs.soliditylang.org/en/v0.8.14/style-guide.html#naming-conventions

Technical Details

YFUtechne.sol

Impact

Informational.

Recommendation

Change variable and function names to mixedCase for readability.

Developer response

Acknowledged

Final remarks

blockdev

The contract is pretty short and avoids complexity. No major risks were found.

NibblerExpress

There is some minor clean-up to be done, but the code otherwise relies on highly used standards that are likely to be secure.

About yAcademy

yAcademy is an ecosystem initiative started by Yearn Finance and its ecosystem partners to bootstrap sustainable and collaborative blockchain security reviews and to nurture aspiring security talent. yAcademy includes a fellowship program, a residents program, and a guest auditor program. In the fellowship program, fellows perform a series of periodic security reviews and presentations during the program. Residents are past fellows who continue to gain experience by performing security reviews of contracts submitted to yAcademy for review (such as this contract). Guest auditors are experts with a track record in the security space who temporarily assist with the review efforts.