Silver Consulting- Fundraiser Finance review

Review Resources:

The project's repository was provided.

Auditor:

Anh Nguyen (Usua◎ Silver)

Table of Contents

Review Summary

Fundraiser Finance

A trustless platform for communities to create charities and donate for their respective cause. Being powered by Ethereum, everything is transparent and trustless. Organizers can set goals in both $ETH and $USDC. By allowing $USDC, we provide the organizers an option to choose non-volatile denominating currency. During the period of collection, all the collected funds will be earning yield from Aave and are held in an escrow. If the goal is met under the collection period, the funds are moved to the organizer's wallet. However, if the goal is not met under the time, then everyone who donated gets an option to withdraw their funds. Upon success of a charity goal, 2% of the fees will be collected by the platform in order to maintain the project. In addition to this, participants will be given an NFT as a badge for their donations.

The dev branch of the Fundraiser Finance Repo was reviewed over 3 days. Concretely, the following file was audited:

- Contracts/Badge.sol
- Contracts/CharityFactory.sol

The contracts were reviewed from October 2 to October 5. The repository was under active development during the review, but the review was limited to one specific commit.

Scope

Code Repo

Commit

The commit reviewed was 61beabbf8e48b67d47a59f0bd686a1f875adbca8. The review covered the entire repository at this specific commit but focused on the contracts/ directory.

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.

Silver Consulting make no warranties regarding the security of the code and do not warrant that the code is free from defects. Silver Consulting does 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, Fundraiser Finance and users agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access controls are applied where needed
Mathematics	Good	Solidity 0.8.13 is used, which provides overflow and underflow protect. Unchecked code was checked in this audit to yield no anomalies. No low-level bitwise operations are performed. There was no unusually complex math.
Complexity	Good	The similar variables names can make the code hard to follow at times.
Libraries	Good	Only basic Open Zeppelin contracts such as IERC20, ERC721, Counters, and Chainlink AggregatorV3Interface are imported, no other external libraries are used. Fewer and simpler external dependencies is always a plus for security.
Code stability	Average	Changes were reviewed at a specific commit and the scope was not expanded after the review was started.
Documentation	Average	Comments existed in natspec but lack clarification about what the code did. Some documentation had to be added to distinguish between variable with similar names, as well as fixing some typos.
Monitoring	Good	Events were added to all important functions that modified state variables.
Testing and verification	Good	All tests were passing after a small fix to a typo, and test coverage was very expansive.

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

High Findings

1. High - Reentrancy vulnerabilities

Proof of concept

```
155     USDC_ADDRESS.transferFrom(address(this), msg.sender, currentDonation.usdcRaised);
156
157     // withdraw eth
158     (bool success,) = payable(msg.sender).call{value: currentDonation.ethRaised}("");
159     require(success, "Failed to transfer eth");
160
161
162     donations[msg.sender][charityId] = UserDonation({
163             ethRaised: 0,
164             usdcRaised: 0
165         });
```

Impact

State variables modified after external call

Recommendation

Modify state variables before calling an external contract.

2. High - Unchecked transfer

2a)

Proof of concept

```
131         USDC_ADDRESS.transferFrom(msg.sender, address(this), amount);
```

Impact

The return value of an external transfer/transferFrom call is not checked

Recommendation

Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

2b)

Proof of concept

```
155     USDC_ADDRESS.transferFrom(address(this), msg.sender, currentDonation.usdcRaised);
```

Impact

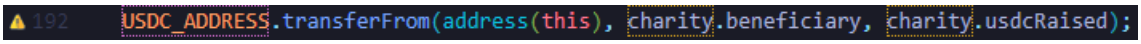
The return value of an external transfer/transferFrom call is not checked

Recommendation

Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

2c)

Proof of concept



Impact

The return value of an external transfer/transferFrom call is not checked

Recommendation

Use SafeERC20 or ensure that the transfer/transferFrom return value is checked.

Medium Findings

1. Medium - Reentrancy vulnerabilities

Proof of concept

```
209     badge.mint(msg.sender, charityId↑, userDonation.ethRaised, userDonation.usdcRaised);
210     nftAlreadyReceived[msg.sender][charityId↑] = true;
211     // solidity does not support null structs
212     donations[msg.sender][charityId↑] = UserDonation({
213             ethRaised: 0,
214             usdcRaised: 0
215         });
```

Impact

State variables modified after external call

Recommendation

Modify state variables before calling an external contract. Apply the check-effects-interactions pattern.

2. Medium - Unused return

2a)

Proof of concept

```
130         USDC_ADDRESS.approve(address(this),  amount↑);
```

Impact

The return value of an external call is not stored in a local or state variable.

Recommendation

Ensure that all the return values of the function calls are used.

2b)

Proof of concept

src/CharityFactory.sol

Impact

The return value of an external call is not stored in a local or state variable.

Recommendation

Ensure that all the return values of the function calls are used.

Low Findings

1. Low - Missing zero address validation

Proof of concept

Detect missing zero address validation.

Impact

Risk of losing ownership of the contract.

```
28    constructor(address _charityFactoryAddress↑) ERC721("FundraiserFinanceParticipationBadge", "FFPB") {
29        charityFactoryAddress = _charityFactoryAddress↑;
30    }
```

Recommendation

Check that the address is not zero.

2. Low - Reentrancy vulnerabilities

Proof of concept

State variables modified after external call, which might act as a double call

Impact

```
136                donations[msg.sender][charityId↑] = UserDonation({
137                    usdcRaised: amount↑,
138                    ethRaised: 0
139                });
140
141            } else {
142                donations[msg.sender][charityId↑].usdcRaised += amount↑;
```

State variables written after external calls

```
⚠ 130            USDC_ADDRESS.approve(address(this),  amount↑);
⚠ 131            USDC_ADDRESS.transferFrom(msg.sender, address(this), amount↑);
```

Recommendation

Apply the [check-effects-interactions pattern].

3. Low - Reentrancy vulnerabilities

Proof of concept

Reentrancies leading to out-of-order events , which might lead to issues for third parties.

Impact

3a)

```
144             emit Contribute(msg.sender, charityId↑, Currency.USDC, amount↑);
```

Event emitted after external calls

```
⚠ 130             USDC_ADDRESS.approve(address(this),  amount↑);
⚠ 131             USDC_ADDRESS.transferFrom(msg.sender, address(this), amount↑);
```

3b)

```
200             emit CloseCharity(charityId↑, charity.status);
```

Event emitted after external calls

```
192             USDC_ADDRESS.transferFrom(address(this), charity.beneficiary, charity.usdcRaised);
```

3c)

```
216             emit ReceiveNtf(msg.sender, charityId↑);
```

Event emitted after external calls

```
⚠ 209             badge.mint(msg.sender, charityId↑, userDonation.ethRaised, userDonation.usdcRaised);
```

3d)

```
166         emit WithdrawContribution(msg.sender, charityId↑, currentDonation.ethRaised, currentDonation.usdcRaised);
```

Event emitted after external calls

```
⚠ 155             USDC_ADDRESS.transferFrom(address(this), msg.sender, currentDonation.usdcRaised);
⚠ 158             (bool success,) = payable(msg.sender).call{value: currentDonation.ethRaised}("");
```

Recommendation

Apply the [check-effects-interactions pattern](#).

4. Low - Block timestamp

Proof of concept

Dangerous usage of `block.timestamp`.`block.timestamp` can be manipulated by miners.

Impact

4a)

```
70          require(block.timestamp < endPeriod, "Charity cannot end in the past");
```

4b)

```
103         require(block.timestamp < charity.endPeriod, "Cannot donate to closed charity");
```

4c)

```
125         require(block.timestamp < charity.endPeriod, "Cannot donate to closed charity");
```

4d)

```
172         require(block.timestamp >= charity.endPeriod, "Cannot close charity until end period pass");
```

Recommendation

Avoid relying on `block.timestamp`.

Informational Findings

1. Informational - Function Initializing State

Proof of concept

Detects the immediate initialization of state variables through function calls that are not pure/constant, or that use non-constant state variable. Special care must be taken when initializing state variables from an immediate function call so as not to incorrectly assume the state is initialized.

Impact

harityDefaultEndTimestamp (test/CharityFactory.t.sol#21) is set pre-construction with a non-constant function or state variable:

- creationTimestamp + 1000

```
20      uint256 creationTimestamp = 1000000;
21      uint256 charityDefaultEndTimestamp = creationTimestamp + 1000;
```

Recommendation

Remove any initialization of state variables via non-constant state variables or function calls. If variables must be set upon contract deployment, locate initialization in the constructor instead.

2. Informational - Incorrect versions of Solidity

Proof of concept

Using an old version prevents access to new Solidity security checks.

Impact

Pragma version^0.8.13 (script/DeployLocal.sol#2) allows old versions

Pragma version^0.8.13 (src/Badge.sol#2) allows old versions

Pragma version^0.8.13 allows old versions at

- (src/CharityFactory.sol#2)
- (test/Badge.t.sol#2)
- (test/CharityFactory.t.sol#2)

Pragma version^0.8.9 (test/mock/MockERC20.sol#2) allows old versions

Recommendation

Use a simple pragma version such as 0.8.10. Consider using the latest version of Solidity for testing.

3. Informational - Low-level calls

Proof of concept

The use of low-level calls is error-prone. Low-level calls do not check for [code existence] or call success.

Impact

src/CharityFactory.sol

```
158         (bool success,) = payable(msg.sender).call{value: currentDonation.ethRaised}("");
195             (bool success,) = payable(charity.beneficiary).call{value: charity.ethRaised}("");
```

Recommendation

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

4. Informational - Conformance to Solidity naming conventions

Proof of concept

Solidity defines a [naming convention] that should be followed.

Impact

- Variable CharityFactory.USDC_ADDRESS (src/CharityFactory.sol#50) is not in mixedCase

```
50      IERC20 private immutable USDC_ADDRESS;
```

- Variable CharityFactoryTest.ETH_MIN_PRICE (test/CharityFactory.t.sol#13) is not in mixedCase
- Variable CharityFactoryTest.ETH_NOT_SUFFICIENT_PRICE (test/CharityFactory.t.sol#14) is not in mixedCase

```
13      uint256 ETH_MIN_PRICE = 0.01 ether;
14      uint256 ETH_NOT_SUFFICIENT_PRICE = 0.009 ether;
```

Recommendation

Follow the Solidity [naming convention].

5. Informational - Variable names too similar

Proof of concept

Detect variables with names that are too similar, which are difficult to read and review.

Impact

Variable CharityFactory.USDC_ADDRESS (src/CharityFactory.sol#50) is too similar to CharityFactory.constructor(address,address)._usdcAddress (src/CharityFactory.sol#62)

```
50        IERC20 private immutable USDC_ADDRESS;
```

```
62        constructor(address _usdcAddress, address _aggregatorAddress) {
```

Recommendation

Prevent variables from having similar names.

6. Informational - Too many digits

Proof of concept

Literals with many digits are difficult to read and review.

Impact

test/CharityFactory.t.sol

```
20        uint256 creationTimestamp = 1000000;
```

Recommendation

Use:

- Ether suffix,
- Time suffix, or
- The scientific notation

Gas Findings

Gas - State variables that could be declared constant

Proof of concept

Constant state variables should be declared constant to save gas.

Impact

test/CharityFactory.t.sol

```
13      uint256 ETH_MIN_PRICE = 0.01 ether;
14      uint256 ETH_NOT_SUFFICIENT_PRICE = 0.009 ether;
```

```
20      uint256 creationTimestamp = 1000000;
```

Recommendation

Add the `constant` attributes to state variables that never change.

Final remarks

With a focus on how contracts interact with each other and vulnerabilities with any external-call mechanics to the main CharityFactory.sol contract, I found nothing particularly worthy of note that was a critical exploit