SECURITY AUDIT REPORT

for project: **Rigor Protocol**

by auditor: **Odd Sequence**

issued: **29 December 2022**

ONE-PAGER

ABOUT

PROJECT

Name	Rigor protocol
Description	Rigor allows running the financing of construction projects and managing relations between contractors and other parties.

AUDITOR

Team	Odd Sequence
Specialists	Two specialists

TASK

Scope links	Github page: - page [link] - commit 937c7c2d6739a46991c6a851bd77304195e668c2 [link]
Scope Description	Smart-contracts Transaction flow Connections with external contracts
Not-in-Scope	IHomeFi.sol IProject.sol IDisputes.sol ICommunity.sol IDebtToken.sol IProjectFactory.sol Best-Practice code remarks Standardized libraries
Networks	EVM networks
Languages	Solidity
Deployment env.	Hardhat framework project
Timeline	bughunting: 01.08.2022 - 06.08.2022
To Do	To find some vulnerabilities of the code in the Scope, that will be possible to find given the time and efforts of specialists.
Done	The auditor has found some vulnerabilities. They are described in the report as “Findings”. The report does not cover all vulnerabilities possible.
Terms	Auditors received a reward from Code4rena after the Rigor Finance contest on Code4rena. Auditors reported findings to Code4rena and received acceptance for some of them.

FINDINGS

6 findings

HIGH acknowledged	Anyone can reduce a project's debt to close-to-zero,using Community.escrow()
HIGH acknowledged	If a past subcontractor has been kicked, he can steal funds from the new subcontractor for task completed
MEDIUM acknowledged	NFTs are transferable, buy Projects are not
MEDIUM acknowledged	Inputs for Project.updateProjectHash() and Project.updateTaskHash() can match different projects
MEDIUM acknowledged	Deployment and initializations for all contracts require two transactions. Frontrun for initialization is possible
LOW acknowledged	Hash for Project is not controlled - projects with the same hash are possible, and NFTs with the same URI

1/2
HIGH

@ Community.sol

Acknowledged

Anyone can reduce a project's debt to close-to-zero, using `Community.escrow()`

Description

Data+signature for previous calls of `Community.escrow()` can be called one more time by anyone, to reduce project's debt.
It breaks all balances significantly between parties.

@ Community.sol, lines 508-552

```
 function escrow(bytes calldata _data, bytes calldata _signature)
        external
        virtual
        override
        whenNotPaused
    {
        // Decode params from _data
        (
            uint256 _communityID,
            address _builder,
            address _lender,
            address _agent,
            address _project,
            uint256 _repayAmount,
            bytes memory _details
        ) = abi.decode(
                _data,
                (uint256, address, address, address, address, uint256, bytes)
            );

        // Compute hash from bytes
        bytes32 _hash = keccak256(_data);

        // Local instance of variable. For saving gas.
        IProject _projectInstance = IProject(_project);

        // Revert if decoded builder is not decoded project's builder
        require(_builder == _projectInstance.builder(), "Community::!Builder");

        // Revert if decoded _communityID's owner is not decoded _lender
        require(
            _lender == _communities[_communityID].owner,
            "Community::!Owner"
        );

        // check signatures
        checkSignatureValidity(_lender, _hash, _signature, 0); // must be lender
        checkSignatureValidity(_builder, _hash, _signature, 1); // must be builder
        checkSignatureValidity(_agent, _hash, _signature, 2); // must be agent or escrow

        // Internal call to reduce debt
        _reduceDebt(_communityID, _project, _repayAmount, _details);
        emit DebtReducedByEscrow(_agent);
    }
```

Lines

[link]

Exploit Scenario

`Community.escrow()` reduces debt for a project. To call the function data+signature is required.
If this transaction happened once, right to reduce debt are in onchain history.

Anyone can call `Community.escrow()` now with these disclosed data+signature to reduce debt more and more.

Recommendation

Add nonce in data or store used data+signature

2/2
HIGH

@ Project.sol

Acknowledged

If a past subcontractor has been kicked, he can steal funds from the new subcontractor for task completed

Description

The issue is with the repeated calls of `Project.changeOrder()` by anyone.
The worse usecase - past subcontractor can steal funds from new subcontractor for task completed.

Subcontractor's funds for completion can be stolen by a previous subcontractor (if any). Bad subcontractor can frontrun `setComplete()` transaction, if he calls `Project.changeOrder()` with already disclosed data+signature, set himself as an active subcontractor and receives funds from task completed.

@ Project.sol, lines 330-359 and lines 386-490

```
function setComplete(bytes calldata _data, bytes calldata _signature)
       external
       override
   {
       // Decode params from _data
       (uint256 _taskID, address _projectAddress) = abi.decode(
           _data,
           (uint256, address)
       );

            // Revert if decoded project address does not match this contract. Indicating incorrect
 _data.
       require(_projectAddress == address(this), "Project::!Project");

       // If the sender is disputes contract, then do not check for signatures.
       if (_msgSender() != disputes) {
           // Check signatures.
           checkSignatureTask(_data, _signature, _taskID);
       }

       // Mark task as complete. Only works when task is active.
       tasks[_taskID].setComplete();

       // Transfer funds to subcontractor.
       currency.safeTransfer(
           tasks[_taskID].subcontractor,
           tasks[_taskID].cost
       );

       emit TaskComplete(_taskID);
   }
...
 function changeOrder(bytes calldata _data, bytes calldata _signature)
        external
        override
        nonReentrant
   {
       // Decode params from _data
       (
           uint256 _taskID,
           address _newSC,
           uint256 _newCost,
           address _project
       ) = abi.decode(_data, (uint256, address, uint256, address));

       // If the sender is disputes contract, then do not check for signatures.
       if (_msgSender() != disputes) {
           // Check for required signatures.
           checkSignatureTask(_data, _signature, _taskID);
       }

       // Revert if decoded project address does not match this contract. Indicating incorrect _data.
       require(_project == address(this), "Project::!projectAddress");

       // Local variable for task cost. For gas saving.
       uint256 _taskCost = tasks[_taskID].cost;

       // Local variable indicating if subcontractor is already unapproved.
       bool _unapproved = false;

       // If task cost is to be changed.
       if (_newCost != _taskCost) {
           // Check new task cost precision. Revert if too precise.
           checkPrecision(_newCost);
```

```
            // Local variable for total cost allocated. For gas saving.
            uint256 _totalAllocated = totalAllocated;

            // If tasks are already allocated with old cost.
            if (tasks[_taskID].alerts[1]) {
                // If new task cost is less than old task cost.
                if (_newCost < _taskCost) {
                    // Find the difference between old - new.
                    uint256 _withdrawDifference = _taskCost - _newCost;

                    // Reduce this difference from total cost allocated.
                    // As the same task is now allocated with lesser cost.
                    totalAllocated -= _withdrawDifference;

                    // Withdraw the difference back to builder's account.
                    // As this additional amount may not be required by the project.
                    autoWithdraw(_withdrawDifference);
                }
                // If new cost is more than task cost but total lent is enough to cover for it.
                else if (totalLent - _totalAllocated >= _newCost - _taskCost) {
                    // Increase the difference of new cost and old cost to total allocated.
                    totalAllocated += _newCost - _taskCost;
                }
                // If new cost is more than task cost and totalLent is not enough.
                else {
                   // Un-confirm SC, mark task as inactive, mark allocated as false, mark lifecycle as
None

                    // Mark task as inactive by unapproving subcontractor.
                    // As subcontractor can only be approved if task is allocated
                    _unapproved = true;
                    tasks[_taskID].unApprove();

                    // Mark task as not allocated.
                    tasks[_taskID].unAllocateFunds();

                    // Reduce total allocation by old task cost.
                    // As as needs to go though funding process again.
                    totalAllocated -= _taskCost;

                    // Add this task to _changeOrderedTask array. These tasks will be allocated first.
                    _changeOrderedTask.push(_taskID);
                }
            }

            // Store new cost for the task
            tasks[_taskID].cost = _newCost;

            emit ChangeOrderFee(_taskID, _newCost);
        }

        // If task subcontractor is to be changed.
        if (_newSC != tasks[_taskID].subcontractor) {
            // If task is not already unapproved, then un-approve it.
            // Un-approving task means marking subcontractor as unconfirmed.
            if (!_unapproved) {
                tasks[_taskID].unApprove();
            }

            // If new subcontractor is not zero address.
            if (_newSC != address(0)) {
                // Invite the new subcontractor for the task.
                _inviteSC(_taskID, _newSC, true);
            }
            // Else store zero address for the task subcontractor.
            // This implies that a subcontractor is not invited from the task.
            else {
                tasks[_taskID].subcontractor = address(0);
            }

            emit ChangeOrderSC(_taskID, _newSC);
        }
    }
```

Lines

[link],[link]

Exploit Scenario

- `setComplete()` is called to pay to new subcontractor for his completed task, transaction is in mempool

	- past subcontractor (previously kicked) picks some `changeOrder()` in history, where he was a subcontractor - past subcontractor frontruns #1, he transact with `changeOrder()` - **past subcontractor is now an active subcontractor** - `setComplete()` happens, bad-subcontractor-frontrunner receives funds **Recommendation** Add `nonce` in data or store used data+signature

1/3
MEDIUM
 	
@ HomeFi.sol
 	
Acknowledged
	NFTs are transferable, buy Projects are not
	
	Description
	Anyone can create a project. Creating a project is equal to mining NFT.
	At the beginning their owners are the same.But NFT means nothing, as with transferring NFT the ownership of the Project is not transferred (there is no way to change the builder for a project).
	It means total dissinchronization between `NFTs` and `Projects`.
	
	@ HomeFi.sol, lines 210-232
	
	`function createProject(bytes memory _hash, address _currency)`
	`external`
	`override`
	`nonReentrant`
	`{`
	`// Revert if currency not supported by HomeFi`
	`validCurrency(_currency);`
	
	`address _sender = _msgSender();`
	
	`// Create a new project Clone and mint a new NFT for it`
	`address _project = projectFactoryInstance.createProject(`
	`_currency,`
	`_sender`
	`);`
	`mintNFT(_sender, string(_hash));`
	
	`// Update project related mappings`
	`projects[projectCount] = _project;`
	`projectTokenId[_project] = projectCount;`
	
	`emit ProjectAdded(projectCount, _project, _sender, _currency, _hash);`
	`}`
	
	Lines
	[link]
	
	Exploit Scenario
	NFT allows transferring as it imports ERC721Upgradeable.sol
	Like in a deployed contract:
	https://rinkeby.etherscan.io/address/0x9361613391Ae3de8112F9571193268363B8704C5#writeProxyContract
	
	But `Project.builder` cannot be changed.
	
	Recommendation
	Disable transfership of change builders NFT is transferred.

2/3
MEDIUM
 	
@ Project.sol
 	
Acknowledged
	Inputs for `Project.updateProjectHash()` and `Project.updateTaskHash()` can match different projects
	
	Description
	It is expected that the data+signature for `updateProjectHash()` will be triggered once and for an exact project.
	But anyone can change a `project hash`, if he will find some easy matching parameters for another project.
	
	@ Project.sol, lines 162-182 and lines 266-293
	
	`function updateProjectHash(bytes calldata _data, bytes calldata _signature)`
	`external`
	`override`
	`{`
	`// Check for required signatures`
	`checkSignature(_data, _signature);`

```
        // Decode params from _data
        (bytes memory _hash, uint256 _nonce) = abi.decode(
            _data,
            (bytes, uint256)
        );

        // Revert if decoded nonce is incorrect. This indicates wrong _data.
        require(_nonce == hashChangeNonce, "Project::!Nonce");

        // Increment to ensure a set of data and signature cannot be re-used.
        hashChangeNonce += 1;

        emit HashUpdated(_hash);
    }
...
 function updateTaskHash(bytes calldata _data, bytes calldata _signature)
        external
        override
    {
        // Decode params from _data
        (bytes memory _taskHash, uint256 _nonce, uint256 _taskID) = abi.decode(
            _data,
            (bytes, uint256, uint256)
        );

        // Revert if decoded nonce is incorrect. This indicates wrong data.
        require(_nonce == hashChangeNonce, "Project::!Nonce");

        // If subcontractor has confirmed then check signature using `checkSignatureTask`.
        // Else check signature using `checkSignature`.
        if (getAlerts(_taskID)[2]) {
            // If subcontractor has confirmed.
            checkSignatureTask(_data, _signature, _taskID);
        } else {
            // If subcontractor not has confirmed.
            checkSignature(_data, _signature);
        }

        // Increment to ensure a set of data and signature cannot be re-used.
        hashChangeNonce += 1;

        emit TaskHashUpdated(_taskID, _taskHash);
    }
```

Lines

[link], [link]

Proof of Concept

Requirements to execute function for different projects and for `Project.updateTaskHash()` with same inputs:

- nonce is the same (they are likely to low or zero for every project)
- both projects don't have contractors but the same builder (to pass `checkSignature()`)
- OR both projects have the same contractor delegated (to pass `checkSignature()`)
- OR both projects have the same pair of builder+contractor (to pass `checkSignature()`)

Recommendation

Add project_address to data like in `Project.changeOrder()`

3/3
MEDIUM

@ HomeFiProxy.sol
@ HomeFi.sol
@ Community.sol
@ Disputes.sol
@ ProjectFactory.sol
@ DebtToken.sol

Acknowledged

Deployment and initializations for all contracts require two transactions. Frontrun for initialization is possible

Description

Anyone can frontrun initialization and feed any inputs. The project will not work.

Some inputs may allow project working, but can steal funds later (like HomeFi.treasury, HomeFi address in all contracts). Attacker can deploy his own contract with the hidden behavior, feed them to `init()` (as if it's a HomeFi contract) and steal funds when they appear on `Community` or `Project` contracts.

Lines

[link], [link], [link], [link], [link], [link]

Proof of Concept

Below the Etherscan addresses with the transactions of contracts. Everywhere we see `initialize()` as a

	method for the first transaction - it means `init()` is a separate transaction after contract deploy. HomeFiProxy.sol - deploy and init by EOA in two transactions https://rinkeby.etherscan.io/address/0x5d00B82009E20E7fa4C1506d3D785702094C3828 HomeFi.sol - deployed by HomeFiProxy and init by EOA in two transactions https://rinkeby.etherscan.io/address/0x9361613391Ae3de8112F9571193268363B8704C5 Community.sol - deployed by HomeFiProxy and init by EOA in two transactions https://rinkeby.etherscan.io/address/0xDd725766c86B0BEe05A89b5587aAd4da3ACE37Ee Disputes.sol - deployed by HomeFiProxy and init by EOA in two transactions https://rinkeby.etherscan.io/address/0x448d745DeA689388CbAd5b1Bea8c204E9464C8C1 ProjectFactory.sol - deployed by HomeFiProxy and init by EOA in two transactions https://rinkeby.etherscan.io/address/0xD2DE87906f78002058587123c31EDC43f94a6Bba DebtToken.sol - deployed by HomeFiProxy and init by EOA in two transactions https://rinkeby.etherscan.io/address/0x023b8c95230D20b8D961E1A3D865329ed68D948a Also proofs: - tests separate transactions too - initiateHomeFi() in HomeFiProxy deploys contracts without init()'s **Recommendation** Options: - Make deployments+init in one transaction via multisig wallet supporting wrapping transactions in calls. - Change HomeFiProxy to deploy all contracts separately. The first transaction for deploy+init of ProjectFactory. The second transaction for deploy+init of Community.sol. And so one. - Place all transactions to flashbots, in one bundle.
1/1 **LOW** **@ HomeFi.sol** **Acknowledged**	**Hash for Project is not controlled - projects with the same hash are possible, and NFTs with the same URI** **Description** Inputed `_hash` for `HomeFi.sol` is the URI for NFTs. Contracts do not check duplicates and do not check if it is correct or not. It is not dangerous within smart-contracts (as NFT is not used much here), but it can be important together with frontend or backend. Hashes will not be unique. **@ HomeFi.sol, lines 210-232 and lines 284-297** `function createProject(bytes memory _hash, address _currency) external override nonReentrant { // Revert if currency not supported by HomeFi validCurrency(_currency); address _sender = _msgSender(); // Create a new project Clone and mint a new NFT for it address _project = projectFactoryInstance.createProject(_currency, _sender); mintNFT(_sender, string(_hash)); // Update project related mappings projects[projectCount] = _project; projectTokenId[_project] = projectCount; emit ProjectAdded(projectCount, _project, _sender, _currency, _hash); }`  `...`  `function mintNFT(address _to, string memory _tokenURI) internal returns (uint256) { // Project count starts from 1 projectCount += 1; // Mints NFT and set token URI _mint(_to, projectCount); _setTokenURI(projectCount, _tokenURI);`

```
        emit NftCreated(projectCount, _to);
        return projectCount;
    }
```

Lines

[link], [link]

Recommendation

Check if it is necessary for NFT hashes to be unique - if so, write checks.

DISCLOSURES

CODE IN THE SCOPE

GITHUB

Link	https://github.com/code-423n4/2022-08-rigor/tree/937c7c2d6739a46991c6a851bd77304195e668c2
Contracts	HomeFi.sol HomeFiProxy.sol Project.sol ProjectFactory.sol Disputes.sol Community.sol DebtToken.sol

REPORT CHANGELOG

First version of findings can be checked via this links:

code-423n4/2022-08-rigor-findings/issues/130
code-423n4/2022-08-rigor-findings/issues/128
code-423n4/2022-08-rigor-findings/issues/139
code-423n4/2022-08-rigor-findings/issues/140
code-423n4/2022-08-rigor-findings/issues/119
code-423n4/2022-08-rigor-findings/issues/120

The later versions are published on Odd Sequence Github page, where changes can be tracked:
https://github.com/oddsequence

FINDINGS STATUS

HIGH

1/1	Acknowledged
2/2	Acknowledged

MEDIUM

1/3	Acknowledged
2/3	Acknowledged
3/3	Acknowledged

LOW

1/1	Acknowledged

DISCLOSURES

GENERAL DISCLOSURES

Odd Sequence typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via Github, website mainpage and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. 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. Odd Sequence owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS

The Reports and the analysis described therein are created solely for Clients and published with their consent. This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any code itself presents unique and unquantifiable risks as the programming languages itself remain under development and is subject to unknown risks and flaws. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

Odd Sequence makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. Odd Sequence hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Odd Sequence. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Odd Sequence is not responsible for the content or operation of such Web sites, and that Odd Sequence shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that Odd Sequence endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. Odd Sequence assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Odd Sequence; however, Odd Sequence does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

NOTICE OF CONFIDENTIALITY

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Odd Sequence. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Odd Sequence. Odd Sequence assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software. In some instances, Odd Sequence may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.