Security Audit Report

RINU INU

22/12/2021

PREPARED FOR:
RINU INU
https://rinu.io/

ARCADIA CONTACT INFO
Email: audits@arcadiamgroup.com
Telegram: https://t.me/thearcadiagroup

THE ARCADIA GROUP

Table of Contents

Executive Summary

A Representative Party of **RINU INU** ("**CLIENT**") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following **RININU/RINUPADV2** smart contracts on the **RINUINU/SMART_CONTRACTS** github repository at Commit #**74b50ea92e8746829d920a5d7fa54a2fcec326b1**.

The scope of this audit included the following files:

1. **RinuPadv2**.sol

Arcadia completed this security review using various methods primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

There were **4** of issues found, **0** of which were deemed to be 'critical', and **0** of which were rated as 'high'.

Severity Rating	Number of Original Occurrences	Number of Remaining Occurrences
CRITICAL	0	0
HIGH	0	0
MEDIUM	0	0
LOW	3	1
INFORMATIONAL	1	0

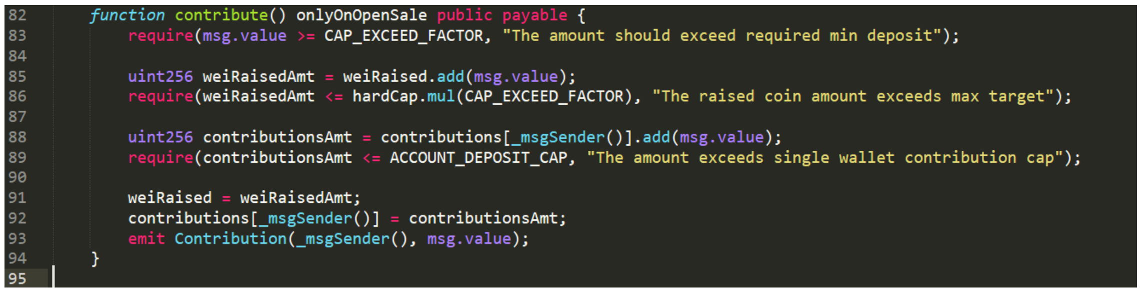
Findings

1. Intermediate Variables

Issue: **RINUPAD-1**
Severity: **LOW**
Likelihood: **LOW**
Impact: **LOW**

Target: **RinuPadv2.sol**
Category: **LOW**
Finding Type: **DYNAMIC**

Both contribute and allocateAndRefund functions use intermediate variables (weiRaisedAmt, contributionsAmt, amount) which are not needed. In fact, these intermediate variables will cause extra gas consumption.

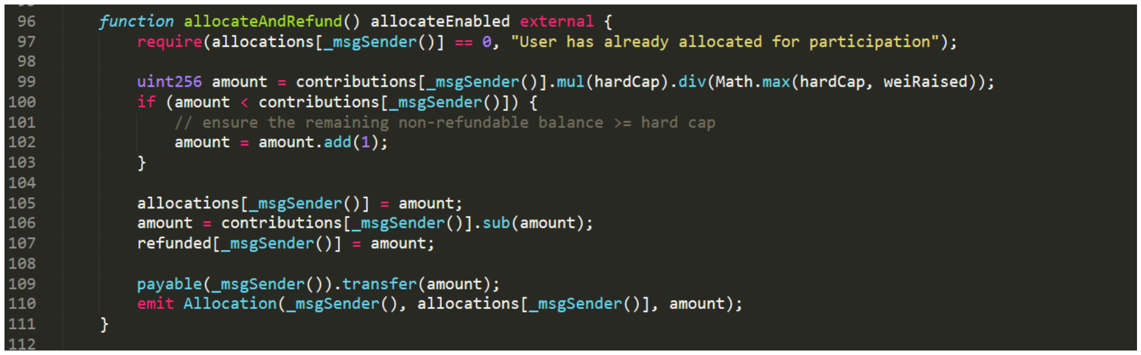


```
    function contribute() onlyOnOpenSale public payable {
        require(msg.value >= CAP_EXCEED_FACTOR, "The amount should exceed required min deposit");

        uint256 weiRaisedAmt = weiRaised.add(msg.value);
        require(weiRaisedAmt <= hardCap.mul(CAP_EXCEED_FACTOR), "The raised coin amount exceeds max target");

        uint256 contributionsAmt = contributions[_msgSender()].add(msg.value);
        require(contributionsAmt <= ACCOUNT_DEPOSIT_CAP, "The amount exceeds single wallet contribution cap");

        weiRaised = weiRaisedAmt;
        contributions[_msgSender()] = contributionsAmt;
        emit Contribution(_msgSender(), msg.value);
    }
```



```
    function allocateAndRefund() allocateEnabled external {
        require(allocations[_msgSender()] == 0, "User has already allocated for participation");

        uint256 amount = contributions[_msgSender()].mul(hardCap).div(Math.max(hardCap, weiRaised));
        if (amount < contributions[_msgSender()]) {
            // ensure the remaining non-refundable balance >= hard cap
            amount = amount.add(1);
        }

        allocations[_msgSender()] = amount;
        amount = contributions[_msgSender()].sub(amount);
        refunded[_msgSender()] = amount;

        payable(_msgSender()).transfer(amount);
        emit Allocation(_msgSender(), allocations[_msgSender()], amount);
    }
```

Action Recommended:

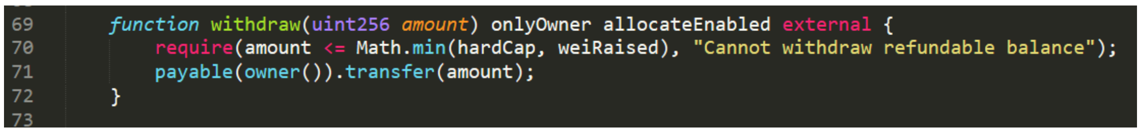
Remove intermediate variables where not needed or compile the contract using Solidity Optimiser. (https://docs.soliditylang.org/en/v0.6.12/internals/optimiser.html)

2. Owner can withdraw refundable balance

Issue: **RINUPAD**-2
Severity: **LOW**
Likelihood: **LOW**
Impact: **HIGH**

Target: **RinuPadv2.sol**
Category: **LOW**
Finding Type: **DYNAMIC**

Even though there’s a requirement check in the amount withdrawn by the owner, he can still withdraw the refundable amount in a second withdraw call.



```
    function withdraw(uint256 amount) onlyOwner allocateEnabled external {
        require(amount <= Math.min(hardCap, weiRaised), "Cannot withdraw refundable balance");
        payable(owner()).transfer(amount);
    }
```

Action Recommended:

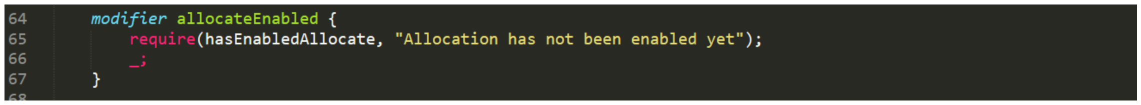
Let withdraw function withdraw once on closed sale the total non-refundable amount: min(hardCap, weiRaised)

3. Allocate enabled modifier

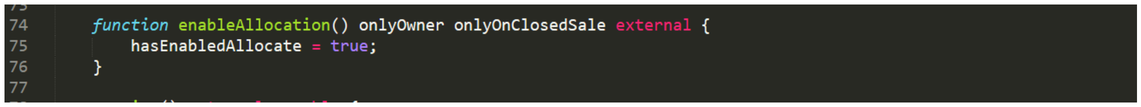
Issue: **RINUPAD**-3
Severity: **LOW**
Likelihood: **LOW**
Impact: **LOW**

Target: **RinuPadv2.sol**
Category: **LOW**
Finding Type: **DYNAMIC**

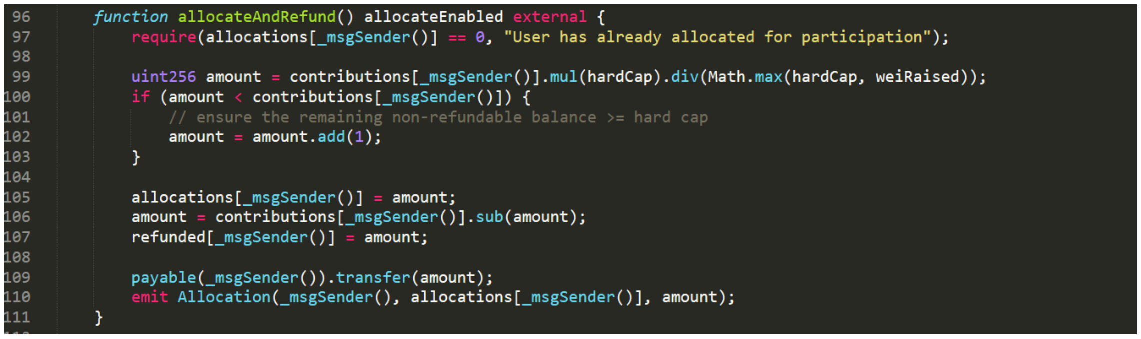
Allocations and refunds are only possible if the owner enables them (on closed sale) whereas contributors shouldn’t be waiting for the owner and should get refunded anytime after the sale closes.



```
    modifier allocateEnabled {
        require(hasEnabledAllocate, "Allocation has not been enabled yet");
        _;
    }
```



```
    function enableAllocation() onlyOwner onlyOnClosedSale external {
        hasEnabledAllocate = true;
    }
```



```
    function allocateAndRefund() allocateEnabled external {
        require(allocations[_msgSender()] == 0, "User has already allocated for participation");

        uint256 amount = contributions[_msgSender()].mul(hardCap).div(Math.max(hardCap, weiRaised));
        if (amount < contributions[_msgSender()]) {
            // ensure the remaining non-refundable balance >= hard cap
            amount = amount.add(1);
        }

        allocations[_msgSender()] = amount;
        amount = contributions[_msgSender()].sub(amount);
        refunded[_msgSender()] = amount;

        payable(_msgSender()).transfer(amount);
        emit Allocation(_msgSender(), allocations[_msgSender()], amount);
    }
```

Action Recommended:

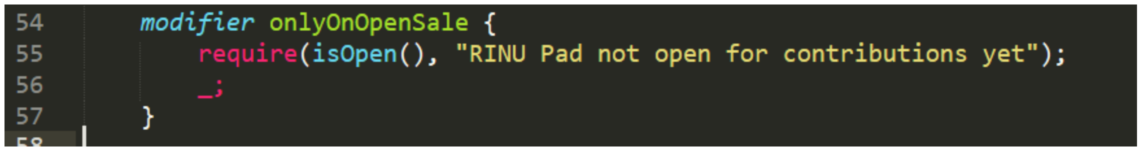
Use onlyOnClosedSale modifier for allocateAndRefund and withdraw functions.

4. onlyOnOpenSale revert message

Issue: **RINUPAD-4**
Severity: **INFO**
Likelihood: **INFO**
Impact: **INFO**

Target: **RinuPadv2.sol**
Category: **INFO**
Finding Type: **DYNAMIC**

This modifier reverts with a confusing message if someone wants to contribute but the sale has closed. The revert message informs that the sale didn't open yet whereas it was opened but did already close.



```
    modifier onlyOnOpenSale {
        require(isOpen(), "RINU Pad not open for contributions yet");
        _;
    }
```

Action Recommended:

Modify reverting message.

Conclusion

Arcadia identified issues that occurred at hash **#74b50ea92e8746829d920a5d7fa54a2fcec326b1** Arcadia also reviewed the fixes for the issues at the following commit: **#7ed68a53085f114414f1bb3e0b717fc6f874b570**

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.