THE ARCADIA GROUP

AUDIT

arcadiamgroup.com

Table of Contents

Executive Summary

A Representative Party of the RampDEFI Protocol ("RampDEFI") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following RampDEFI smart contracts on the RampDEFI repo at Commit #3490b92a0eb03eb3619b82943b576017fd6f3156

RampVault.sol

Controller.sol

SushiLPStrategy.sol

RampStakingStrategy.sol

RUSD.sol

This commit is obviously a work in progress and not a fully functional version of the protocol so that we checked for existing vulnerabilities and mentioned issues that could arise, anyway since the full mechanism was not yet in place in this commit, we pointed out some missing features that were not in the comments like other parts where it was clearly stated that was a TODO feature and not a missing one.

Due to the unfinished state of the commit, for security reasons we recommend to apply the proper changes and review them, but also to audit again once all features are in place and the code is tested and finalized.

Because of this reason we decided not to flag as Critical issues but flag them as High, since it's a work in progress, we expect to review them and update the table below.

NOTE: This audit shows issues that closely affect the overall logic of the protocol if not completed or updated. Plus some functions are declared but not coded as claim() and claimFor() in RampVault.sol

Severity Rating	Number Of Original Occurrences	Number Of Remaining Occurrences
Critical	0	0
High	4	0
Medium	2	0
Low	1	0
Notice	0	0
Informational	1	0

Findings

1. Function _repay does not update any balances

CODE-1
Severity: High
Impact: High

Contract: RampVault.sol
Category: Security
Finding Type: Dynamic
Lines: 190

This internal function doesn't update the borrowed amount and other related balances as expected, but it does burn the rUSD.

```
function _repay(address _token, address _account, uint256 _amount,
uint256 _price)
   internal
   {
       TokenInfo storage token = tokens[_token];

       // Calculate repayable

       uint256 maxRepayable = _maxRepayable(_token, _account, _price);

       // Adjust amount to max if it's more than possible
       if (_amount > maxRepayable) _amount = maxRepayable;

       // Burn the rUSD
       rUSD.burn(_account, _amount);

       // Emit event
       emit Repay(_token, _account, _amount, _price);
   }
```

2. Function _withdraw is missing important logic checks for the withdrawing amount.

CODE-2
Severity: High
Impact: High

Contract: RampVault.sol
Category:
Finding Type: Dynamic
Lines: 307 and 331

In _withdraw function once the maxWithdrawable amount is retrieved, it does not make any security check on that amount. Also balances should be updated before calling the withdraw function. Even if it's a known function, it's always better to update them before to avoid unexpected behaviours when more advanced features will be added.

```
function _withdraw(address _token, address _account, uint256 _amount,
uint256 _price) internal {
        TokenInfo storage tokenInfo = tokens[_token];

        // Get balance
        uint256 balance = _getBalance(_token, _account);

        // Check how much can be withdrawn within collateralization ratio
        uint256 maxWithdrawable = _maxWithdrawable(_token, _account,
_price);

        // TODO revert if withdraw amount too high

        // Depending on the StrategyType, deposits go to Strategy or
Vault
        address withdrawFrom = (tokens[_token].strategy.getStrategyType()
== IStrategy.StrategyTypes.Direct)
        ? address(tokens[_token].strategy)
        : address(this);

        if (tokens[_token].strategy.getStrategyType() ==
```

```
IStrategy.StrategyTypes.Direct) {
            // Direct: Withdraw from strategy
            tokenInfo.strategy.withdraw(_token, _account, _amount);
        } else {
            // TODO Aggregate
        }

        // adjust local balance
        tokenInfo.depositedBalances[msg.sender] =
tokenInfo.depositedBalances[msg.sender].sub(_amount);

        // Emit event
        emit Withdraw(_token, _account, _amount, _price);
    }
```

3. Return value from _maxWithdrawable could be misinterpreted

CODE-3
Severity: Medium
Impact: Medium

Contract: RampVault.sol
Category: Security
Finding Type: Dynamic
Lines: 430

The function returns the maxWithdrawble value for a given _token address, this value does not take into account the collateralRatio for the borrowed amount if any, if it's an expected behavior, consider renaming it with a more specific name, while if this is the actual amount withdrawable then additional logic is required(checks with collateralRatio) before the value is returned.

```
function _maxWithdrawable(address _token, address _account, uint256
_price) internal view returns (uint256) {
        TokenInfo storage token = tokens[_token];
```

```
        // Retrieve amount of rUsd borrowed
        uint256 borrowed = token.borrowedBalances[_account];
        console.log("borrowed", borrowed.div(10 ** 18));

        // Retrieve collateralizationRatio
        uint256 collateralRatio = token.collateralRatio;
        console.log("collateralRatio", collateralRatio.div(10 ** 18));

        // What is the value of the deposited balance
        uint256 depositedUsdValue =
token.depositedBalances[_account].mul(_price).div(10 **
PriceReceiver.PRICE_DECIMALS);
        console.log("depositedUsdValue", depositedUsdValue.div(10 **
18));

        // What usd amount is available
        uint256 usdAvailable = depositedUsdValue.sub(borrowed);

        console.log("usdAvailable", usdAvailable.div(10 ** 18));

        uint256 withdrawableAsset = usdAvailable.div(_price).mul(10 **
PriceReceiver.PRICE_DECIMALS);
        console.log("withdrawableAsset", withdrawableAsset.div(10 **
18));

        return withdrawableAsset;
    }
```

4. Mappings are unused

CODE-4
Severity: None
Impact: None

Contract: Controller.sol
Category: Informational
Finding Type: Dynamic
Lines: 25, 26, 29

In Controller.sol some mappings are unused. The ones which are not used are isVault, isStrategy and converters. Consider remove them.

```
    // Vault to strategy mapping
    mapping(address => address) public vaults;
    // Strategy to vault mapping
    mapping(address => address) public strategies;

    mapping(address => mapping(address => address)) public converters;
    mapping(address => mapping(address => bool)) public
approvedStrategies;

    mapping(address => bool) public isVault;
    mapping(address => bool) public isStrategy;
```

5. Function deposit doesn't deposit tokens but update balances

CODE-5
Severity: High
Impact: High

Contract: SushiLPStrategy.sol
Category: Security
Finding Type: Dynamic
Lines: 106

Balances are updated without requiring any tokens transfer inside the contract. Probably due to the unfinished stage but critical from a security point of view, since it breaks all the logic.

```
    // Deposit LP tokens to MasterChef for SUSHI allocation.
   function deposit(address token, address _account, uint256 _amount)
external override onlyVault {
    uint256 _pid = tokenPoolInfo[token];

    address lpToken = IMasterChef(masterChef).poolInfo(_pid).lpToken;
    require(lpToken == token, "MasterChef lptoken address is
different");

    uint256 accSushiPerShare =
IMasterChef(masterChef).poolInfo(_pid).accSushiPerShare;
    IERC20Upgradeable token = IERC20Upgradeable(token);

    UserInfo storage user = userInfo[_pid][msg.sender];

    // This is the very first deposit
    if (user.amount == 0) {
        user.rewardDebtAtBlock = block.number;
    }

    user.amount = user.amount.add(_amount);
    user.rewardDebt = user.amount.mul(accSushiPerShare).div(UNITS);

    emit Deposit(msg.sender, _pid, _amount);
```

```
    }
```

6. Pid is not defined

CODE-6
Severity: Medium
Impact: Medium

Contract: SushiLPStrategy.sol
Category: Security
Finding Type: Dynamic
Lines: 130

In function withdraw() the pid variable is initiated but it's not defined, and then is used as argument in harvest() within the function. This will always use pid = 0 since it has not been retrieved from the corresponding struct. Proper use of pid should be done like this `uint256 _pid = tokenPoolInfo[_token];`

```
  function withdraw(address _account, address token, uint256 _amount)
external override onlyVault {
      uint256 _pid;
      // check pid is correct
      UserInfo storage user = userInfo[_pid][msg.sender];
      address lpToken = IMasterChef(masterChef).poolInfo(_pid).lpToken;
      IERC20Upgradeable token = IERC20Upgradeable(lpToken);

      require(user.amount >= _amount, "Cannot withdraw more than
balance");

      // Withdraw the underlying tokens from masterChef.
      IMasterChef(masterChef).withdraw(_pid, _amount);
```

```
        // withdraw lp funds back to user
        token.safeTransferFrom(
            address(this),
            address(msg.sender),
            _amount
        );

        _harvest(_pid);
        emit Withdraw(msg.sender, _pid, _amount);
    }
```

7. Fee is calculated and applied but not transferred

CODE-7
Severity: High
Impact: High

Contract: SushiLPStrategy.sol
Category: Security
Finding Type: Dynamic
Lines: 170

In _harvest function the fee is calculated properly and stored in a variable but then is not transferred nor burnt, if the goal is to burn it then consider the possibility of adding a function that can burn the accumulatedFee whenever required while if it has to be transferred, this part is missing.

```
function _harvest(uint256 _pid) internal {
        UserInfo storage user = userInfo[_pid][msg.sender];
```

```
        uint256 accSushiPerShare =
IMasterChef(masterChef).poolInfo(_pid).accSushiPerShare;
        IERC20Upgradeable sushi = IERC20Upgradeable(sushiToken);

        if (user.amount == 0) return;

        uint256 pending =
user.amount.mul(accSushiPerShare).div(UNITS).sub(user.rewardDebt);

        uint256 available = sushi.balanceOf(address(this));

        if (pending > available) {
            pending = available;
        }

        if (pending > 0) {
            uint256 fee = pending.mul(feePercentage).div(100);

            // Burn the fees if there were any
            accumulatedFee = accumulatedFee.add(fee);

            // transfer sushi to user
            sushi.transfer(msg.sender, pending.sub(fee));

            user.rewardDebtAtBlock = block.number;

            emit SendSushiReward(msg.sender, pending.sub(fee));
        }

        user.rewardDebt = user.amount.mul(accSushiPerShare).div(UNITS);

    }
```

8. Balances should be updated before transfer tokens from sushi

CODE-8
Severity: Low
Impact: Low

Contract: SushiLpStrategy.sol
Category: Security
Finding Type: Dynamic
Lines: 194 and 199

Balances for pending calculation are updated only after sushi transfer. Even if the call is made to a known contract and it's a token transfer and not eth, consider updating them before transferring to avoid any possible issue in the future.

```
 function _harvest(uint256 _pid) internal {
       UserInfo storage user = userInfo[_pid][msg.sender];
       uint256 accSushiPerShare =
IMasterChef(masterChef).poolInfo(_pid).accSushiPerShare;
       IERC20Upgradeable sushi = IERC20Upgradeable(sushiToken);

       if (user.amount == 0) return;

       uint256 pending =
user.amount.mul(accSushiPerShare).div(UNITS).sub(user.rewardDebt);

       uint256 available = sushi.balanceOf(address(this));

       if (pending > available) {
           pending = available;
       }

       if (pending > 0) {
           uint256 fee = pending.mul(feePercentage).div(100);

           // Burn the fees if there were any
           accumulatedFee = accumulatedFee.add(fee);
```

```
            // transfer sushi to user
            sushi.transfer(msg.sender, pending.sub(fee));

            user.rewardDebtAtBlock = block.number;

            emit SendSushiReward(msg.sender, pending.sub(fee));
        }

        user.rewardDebt = user.amount.mul(accSushiPerShare).div(UNITS);

    }
```

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.

Document Information

Title	RampDEFI Protocol Audit
Client	RampDEFI
Auditor(s)	Andrea Burzi
Reviewed by	Joel Farris
Approved by	Rasikh Morani
Contact Details	Rasikh Morani (972) 543-3886 rasikh@arcadiamgroup.com https://t.me/thearcadiagroup