www.securing.biz

Smart Contracts Security Review for Unbound

Retest

document version: 1.1

author: Paweł Kuryłowicz (SecuRing)

test time period: 2021-09-27 – 2021-10-14

report date: 2021-10-21

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

1.1. Summary of test results

- There are no vulnerabilities with **critical** or **high** impact on risk found.
- There is one vulnerability with **medium** impact on risk found. Their potential consequences are:
  - Gaining access to functions that have strong influence on the protocol (4.1).
- There are two vulnerabilities with **low** impact on risk found. Their potential consequences are:
  - Access to the function with just one out of two required roles (4.2).
  - Change the value of the CR parameter to any value (4.3).
- Additionally, **nine** recommendations have been proposed that do not have any direct risk impact. However, it is suggested to implement them due to good security practices.

1.2. Summary of retest results

- Only fixes for previously detected vulnerabilities were checked during the retests.
- All vulnerabilities have been fixed.
- 5 out of 9 security recommendations have been implemented.

2. PROJECT OVERVIEW

2.1. Project description

The analyzed Unbound Core contracts act as treasury for liquidity pool tokens. It distinguishes roles such as *User*, *Manager* and *Governance*:

- ***Governance*** - role with the highest privileges responsible for changing parameters, such as fees and addresses, managing the whitelists (e.g. yield factory).
- ***Manager*** – role that is permitted to change selected parameters in the protocol,
- ***User*** – role that locks their LP tokens to get uTokens in return and has a possibility to use their locked LP tokens for yielding.

An example usage scenario:

1. *User* deposit LP tokens (e.g. through *lock* function in UniswapV2Vault contract and specifies farming address).
2. Unbound transfers LP tokens to vault and then to the farm.
3. Unbound mints uTokens according to the current LTV parameter.
4. *User* withdraws their LP tokens and gets the yielding reward.
5. Unbound withdraws LT tokens from yielding contract.
6. Unbound sends to *User* specific amount of collateral according to the current CR parameter.

2.2. Target in scope

The object being analysed were selected smart contracts accessible in the following:

- GitHub repository: https://github.com/unbound-finance/unbound-core
- Commit ID:
  - *f497dc1734135146c04fb053288fe31ae73298a7*
- Additional commit ID used to test farming contracts:
  - *b44aa08ec6857c83ca2b391fae6889c700b39558*

The contracts reviewed were the following:

- *base/*
  - *UnboundVaultBase*
  - *UnboundVaultManager*
- *defiedge/*
  - *DefiEdgeVault*
  - *DefiEdgeVaultFactory*
- *uniswap-v2/*
  - *UniswapV2Vault*
  - *UniswapV2VaultFactory*

- *libraries/*
  - *DefiEdgeSharePriceProvider*
  - *UniswapV2PriceProvider*
- *farming/*
  - *kyberdmm-yield/*
    - *KyberYieldWallet*
    - *KyberYieldWalletFactory*
  - *sushiswap/*
    - *SushiSwapYieldWallet*
    - *SushiSwapYieldWalletFactory*
- *UnboundToken*

2.3. Threat analysis

This section summarized the potential threats that were identified during initial threat modeling. The audit was mainly focused on, but not limited to, finding security issues that be exploited to achieve these threats.

The key threats were identified from particular perspectives as follows:

1. General (apply to all of the roles mentioned)
   - Bypassing the business logic of the system.
   - Theft of users' funds.
   - Errors in arithmetic operation.
   - Incorrect access control for roles used by the contracts.
   - Existing of known vulnerabilities (e.g. front-running, re-entrancy, overflows)
2. Governance
   - Too much power in relation to the declared one.
   - Unintentional loss of the ability to governance the system.
3. User
   - Theft of locked LP tokens.
   - Withdrawal of more funds than expected by users.
   - Earning a higher reward than according to the system's assumptions.

2.4. Testing overview

The security review of the *Unbound* smart contracts were meant to verify whether the proper security mechanisms were in place to prevent users from abusing the contracts' business logic and to detect the vulnerabilities which could cause financial losses to the client or its customers.

Security tests were performed using the following methods:

- Source code review,

- Automated tests through various tools (static analysis),
- Q&A sessions with the client's representatives which allowed to gain knowledge about the project and the technical details behind the platform.

2.5. Basic information

Testing team	Damian Rusinek Paweł Kuryłowicz Jakub Zmyslowski
Testing time period	2021-09-27 – 2021-10-14
Report date	2021-10-14 – 2021-10-15
Retest time period	2021-10-20 – 2021-10-21
Version	1.1

2.6. Disclaimer

The security review described in this report is not a security warranty.
It does not guarantee the security or correctness of reviewed smart contracts. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best security practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we strongly recommend implementing security mechanisms at all stages of development and maintenance.

3. SUMMARY OF IDENTIFIED VULNERABILITIES

3.1. Risk classification

Vulnerabilities	
Risk impact	**Description**
Critical	Vulnerabilities that affect the security of the entire system, all users or can lead to a significant financial loss (e.g. tokens). It is recommended to take immediate mitigating actions or limit the possibility of vulnerability exploitation.
High	Vulnerabilities that can lead to escalation of privileges, a denial of service or significant business logic abuse. It is recommended to take mitigating actions as soon as possible.
Medium	Vulnerabilities that affect the security of more than one user or the system, but might require specific privileges, or custom prerequisites. The mitigating actions should be taken after eliminating the vulnerabilities with critical and high risk impact.
Low	Vulnerabilities that affect the security of individual users or require strong custom prerequisites. The mitigating actions should be taken after eliminating the vulnerabilities with critical, high, and medium risk impact.
Recommendations	
Methods of increasing the security of the system by implementing good security practices or eliminating weaknesses. No direct risk impact has been identified. The decision whether to take mitigating actions should be made by the client.	

3.2. Identified vulnerabilities

Vulnerability	Risk impact	Retest 2021-10-21
4.1 Centralized governance with excessive permissions	**Medium**	**Fixed**
4.2 Modifier name *governanceAndManager* inconsistent with its logic	**Low**	**Fixed**
4.3 The function *changeCR* does not enforce the correct limits	**Low**	**Fixed**
Recommendations		
5.1 Improve *distrubuteFee* function		**Implemented**
5.2 Use specific Solidity compiler version		**Not implemented**
5.3 Consider adding zero-address validation of the addresses		**Implemented**
5.4 Rename functions to better suit its purpose		**Not implemented**
5.5 Make variables self-explanatory		**Implemented**
5.6 Functions does not emit proper events		**Implemented**
5.7 Gas optimization		**Not implemented**
5.8 Add interface inheritance		**Not implemented**
5.9 Add parameter validation		**Implemented**

4. VULNERABILITIES

4.1. Centralized governance with excessive permissions

Risk impact	Medium	SCSVS V1
Vulnerable contract	*UnboundVaultManager*, *UnboundToken*	
Exploitation conditions	Taking over the governance EOA by compromising the private key.	
Exploitation results	Strong influence on the protocol e.g. possibility to change *uToken* minting limit, staking contract address, enable/disable yield wallet factories, add new minters and locking access to user's funds.	
Remediation	*Centralized governance* Using DAO as a source of decisions, introducing predetermined maximum and minimum values where possible and a clear message in the documentation regarding the privileges of each role. Additionally, the use of Timelock may be considered so that users have time to become familiar with the decision and the change that will be made. However, it should be taken into account that in the event of a mistake or vulnerability, it may also be delayed to repair it. *Excessive permissions* Remove *whenNotPaused* modifier from burn function. Change *validMinter* so that minter is disabled a few days after the call.	

Retest 2021-10-21:
Fixed

Centralized governance

- The use of *DAO* as a source of decisions is considered for the future.
- *Timelock* has been added to the *UniswapV2VaultFactory*.
  - Disabling vaults – *governance* may disable particular vault 7 days after calling the *disableVault* function.

  CommitID: f0c1e5726f489bf736e5a7f07e3c53db1de17d04
- *Timelock* has been added to the *DefiEdgeVaultFactory*.
  - Disabling vaults – *governance* may disable particular vault 7 days after calling the *disableVault* function.

  - Enabling vaults – *governance* may enable particular vault 3 days after calling the *enableVault* function.

  CommitID: 8c1f4cd9128b68c9742295eb5ab491401b3d0505
- The team has declared that they will use multisig for *governance.*

Excessive permissions

- *whenNotPaused* modifier was removed from *burn* function.
  CommitID: dd53257d1ca93d8baa5d50aa69866e5ab3d0c5d5

Vulnerability description:
Currently, the project is very much dependent on *Governance*, which is not ensured to be a decentralized contract. Other contracts contain many functions that strongly influence the operation of the project. These decisions take effect immediately after calling them.
Note: There is no governance contract in the repository and deployment script so the safe assumption is that the EOA or multisig contract is used as governance.

Test case:

Centralized governance
Functions that have strong influence on the project and are managed by governance address:

- UnboundVaultManager.sol
  - *changeUTokenMintLimit*
  - *changeTeamFeeAddress*
  - *changeFee*
  - *changeStaking*
  - *changeSafuShare*
  - *changeSafuAddress*
  - *enableYieldWalletFactory*
  - *disableYieldWalletFactory*
- DefiEdgeVaultFactory.sol, UniswapV2VaultFactory.sol
  - *enableVault*
  - *disableVault*
- UnboundToken.sol
  - *addMinter*
  - *enableMinter*

Excessive permissions
The current implementation introduces the risk of blocking the user's funds in the event of compromise or loss of the private key.

- UnboundToken.sol#L70-71

```
function burn(address _account, uint256 _amount)
    external
    whenNotPaused
    validMinter
{
    _burn(_account, _amount);
}
```

References:
SCSVS V1: Architecture, design and threat modeling
https://securing.github.io/SCSVS/1.2/0x10-V1-Architecture-Design-Threat-modelling.html

4.2. Modifier name *governanceAndManager* inconsistent with its logic

Risk impact	Low	SCSVS V8
Vulnerable contract	*UnboundVaultManager*	
Exploitation conditions	Access to *governance* or *manager* role and an instance of a function that requires both of these roles for authorization.	
Exploitation results	Access to the function with just one out of two required roles.	
Remediation	Change modifier name to *governanceOrManager*.	

Retest 2021-10-20:
Fixed

- Modifier name was changed to *governanceOrManager*.
  CommitID: f2c4f065d405ccea42e80adf30bcd4798e745ad9

Vulnerability description:
Modifier name *governanceAndManager* is inconsistent with its logic.

Test case:
The name directly indicates the requirement to have two roles. Inside the implementation, however, we can see that only one of these roles is enough to fulfil the condition.

- UnboundVaultManager.sol#L74-77

```
    modifier governanceAndManager() {
        require(msg.sender == manager || msg.sender == governance, 'NA');
        _;
    }
```

References:
SCSVS V8: Business logic
https://securing.github.io/SCSVS/1.1/0x17-V8-Business-Logic.html

4.3. The function *changeCR* does not enforce the correct limits

Risk impact	Low	SCSVS V8
Vulnerable contract	*UnboundVaultManager*	
Exploitation conditions	Access to *governance* or *manager* role.	
Exploitation results	Change the value of the *CR* parameter to any value.	

Remediation	Add proper require statement to enforce limits correctly:
	`function changeCR(uint256 _CR) external governanceAndManager {`
	`    require(_CR <= SECOND_BASE);`
	`    CR = _CR;`
	`    emit ChangeCR(_CR);`
	`}`	

Retest 2021-10-20:
Fixed

- Proper require statement has been added to *changeCR* function
  CommitID: 4d508af0e068f2ee447bc03e0b519d67c32d83dd

Vulnerability description:
The assigned value of the *CR* parameter is not properly verified.

Test case:
The maximum value of the *CR* parameter, according to the contract comments, should be 1e8 (same as *SECOND_BASE*), but it is not enforced.

- UnboundVaultManager.sol#L105-108

```
    function changeCR(uint256 _CR) external governanceAndManager {
        CR = _CR;
        emit ChangeCR(_CR);
    }
```

References:
SCSVS V8: Business logic
https://securing.github.io/SCSVS/1.1/0x17-V8-Business-Logic.html

5. RECOMMENDATIONS

5.1. Improve *distrubuteFee* function

Retest 2021-10-21:

Implemented

- The comment has been changed to:

```
// check if safu is initialized properly
```

CommitID: 67a03508987899ca0317eeb2218b5d6a28cca722

- The *distributeFee* function was optimized in *else* statement:

```
uToken.transfer(safu, amount);
```

CommitID: 5a6005622d6b99a184b11a30b842c014ed00b05e

Description:

The fee distribution function is inefficient. It allows to distribute 100% of *safuShare* to *safu* address in multiple calls, even if the share is set to 20% but the team address has not yet been set.

- UnboundVaultManager.sol#L205

```
function distributeFee() external {
        // check if safu and team is initialized properly
        require((safu != address(0)) && (safuShare > 0), 'INVALID');
        uint256 amount = uToken.balanceOf(address(this));

        if (team != address(0)) {
            // transfer to safu
            uToken.transfer(safu, amount.mul(safuShare).div(SECOND_BASE));

            // transfer remaining to team
            uToken.transfer(
                team,
                amount.sub(amount.mul(safuShare).div(SECOND_BASE))
            );
        } else {
            // transfer the whole to safu
            uToken.transfer(safu, amount.mul(safuShare).div(SECOND_BASE));
        }
    }
```

How to implement:

When team address is not set, distribute 100% to the *safu* address - do not multiply by their share.

Moreover, the comment at the beginning of the function indicates that team initialization verification needs to be done, which is not done. The comment should be changed to:

```
//check if safu is initialized.
```

References:

SCSVS V8: Business logic

https://securing.github.io/SCSVS/1.2/0x17-V8-Business-Logic.html

5.2. Use specific Solidity compiler version

Retest 2021-10-20:
Not implemented

Description:
Audited contracts use the following pragma:

```
pragma solidity >=0.7.6;
```

It allows to compile contracts with old versions of compiler and to recent versions. Also, use the latest stable version (0.8.x) for testing.

How to implement:
Use a specific version of Solidity compiler (latest stable):

```
pragma solidity 0.7.6;
```

References:
Floating Pragma
https://swcregistry.io/docs/SWC-103
SCSVS V1: Architecture, design and threat modelling
https://securing.github.io/SCSVS/1.1/0x10-V1-Architecture-Design-Threat-modelling.html

5.3. Consider adding zero-address validation of the addresses

Retest 2021-10-21:
Implemented

- A modifier *validAddress* has been created and it is added to all indicated functions *CommitID: 67a03508987899ca0317eeb2218b5d6a28cca722*

Description:
There are a few functions in the contracts that allow to change the address and do not check whether the given address is 0:

- UnboundVaultManager.sol
  - *changeManager*
  - *changeGovernance*
  - *changeStaking*
  - *changeTeamFeeAddress*
- DefiEdgeVaultFactory.sol
  - *changeGovernance*
- UniswapV2VaultFactory.sol
  - *changeGovernance*

They do not check whether the given address is 0. It is good practice to extract separate functions to set the address to zero if it is required for the proper functioning of the project.

How to implement:
Add require statement to verify if the input parameter is not address(0).

References:
SCSVS V1: Architecture, design and threat modelling
https://securing.github.io/SCSVS/1.1/0x10-V1-Architecture-Design-Threat-modelling.html

5.4. Rename functions to better suit its purpose

Retest 2021-10-20:

Not implemented

Description:
In accordance with Solidity's best practices, names should best fit their purpose.

How to implement:
Rename *changeFee* mapping to *changeProtocolFee.*
Rename *changeStaking* mapping to *changeStakeFeeAddress.*

References:
SCSVS V11: Code clarity
https://securing.github.io/SCSVS/1.1/0x20-V11-Code-Clarity.html
Solidity Style Guide page 220
https://docs.soliditylang.org/_/downloads/en/v0.7.6/pdf/

5.5. Make variables self-explanatory

Retest 2021-10-21:

Implemented

- The following comments have been added:

```
    uint256 BASE = uint256(1e18); // used to normalise the value to 1e18 format
    uint256 SECOND_BASE = uint256(1e8); // act as base decimals for LTV and CR
```

CommitID: dcffc2134f789c16be07fac9a6e83b9747802679

Description:
In accordance with Solidity's best practices, variables names should be self-explanatory and have appropriate comments.

How to implement:
Change names of variables to make them self-explanatory and add comments:

```
uint256 BASE = uint256(1e18);
uint256 SECOND_BASE = uint256(1e8);
```

References:
SCSVS V11: Code clarity
https://securing.github.io/SCSVS/1.1/0x20-V11-Code-Clarity.html

5.6. Functions does not emit proper events

Retest 2021-10-21:

Implemented

- Indicated events have been added.

CommitID: dda5f71e71987f9b450b8877a4c248bf81dc02d1

Description:
The following functions does not emit proper events:

- UnboundToken.sol
  - *acceptGovernance*
- UnboundVaultManager.sol
  - *claim*
  - *changeTeamFeeAddress*
  - *changeStaking*
  - *distributeFee*
- UniswapV2VaultFactory.sol
  - *acceptGovernance*
- DefiEdgeVaultFactory.sol
  - *acceptGovernance*
- SushiSwapYieldWallet.sol
  - *claim*
  - *deposit*
- SushiSwapYieldWalletFactory.sol
  - setPids

How to implement:
Emit proper events like in the example below:

```
function acceptGovernance() external {
        require(msg.sender == pendingGovernance, 'NA');
        governance = pendingGovernance;
        emit ChangedGovernance(pendingGovernance);
```

References:
SCSVS V1: Architecture, design and threat modeling
https://securing.github.io/SCSVS/1.1/0x20-V11-Code-Clarity.html

5.7. Gas optimization

Retest 2021-10-20:

Not implemented

Description:
The *hasPriceDifference* function in the *UniswapV2PriceProvider* checks *_reserve0/_reserve1 and _reserve1/_reserve0* redundantly.

```
function hasPriceDifference(
        uint256 _reserve0,
        uint256 _reserve1,
        uint256 _maxPercentDiff
    ) internal pure returns (bool result) {
        uint256 diff = _reserve0.mul(BASE).div(_reserve1);
        if (
            diff > (BASE.add(_maxPercentDiff)) ||
            diff < (BASE.sub(_maxPercentDiff))
        ) {
            return true;
        }
        diff = _reserve1.mul(BASE).div(_reserve0);
        if (
            diff > (BASE.add(_maxPercentDiff)) ||
            diff < (BASE.sub(_maxPercentDiff))
        ) {
            return true;
```

How to implement:
Remove one *diff* calculation and *if* statement in order to reduce gas cost.

References:
SCSVS V7: Gas usage & limitations
https://securing.github.io/SCSVS/1.2/0x10-V1-Architecture-Design-Threat-modelling.html

5.8. Add interface inheritance

Retest 2021-10-20:

Not implemented

Description:
The following contracts does not inherit from interfaces:

- *UnboundToken* contract should inherit from *IUnboundVaultFactory*,
- *UniswapV2VaultFactory* contract should inherit from *IUnboundVaultFactory*.

How to implement:
Inheritance will allow to find possible non-compliance when changes in code are introduced.

References:
SCSVS V1: Architecture, design and threat modelling
https://securing.github.io/SCSVS/1.1/0x10-V1-Architecture-Design-Threat-modelling.html

5.9. Add parameter validation

Retest 2021-10-21:

Implemented

- *DefiEdgeVault* and *UniswapV2Vault* now check the decimals of strategy
  CommitID: 16c0ba89924101f1188249b0caa93e531892a546

Description:
In DefiEdgeVault and UniswapV2Vault contracts the vaults are supposed to handle only 18 decimals LPs, but it is not verified.

How to implement:
Consider checking the decimals of strategy.
For DefiEdgeVault.sol:

```
require(strategy.decimals() == 18, 'Vault: LP decimals not 18');
```

For UniswapV2Vault.sol:

```
require(pair.decimals() == 18, 'Vault: LP decimals not 18');
```

References:
SCSVS V5: Arithmetic
https://securing.github.io/SCSVS/1.2/0x14-V5-Arithmetic.html

6. TERMINOLOGY

This section explains the terms that are related to the methodology used in this report.

Risk = Threat + Vulnerability

Threat

Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.[1]

Vulnerability

Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.[1]

Risk

The level of impact on organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals resulting from the operation of an information system given the potential impact of a threat and the likelihood of that threat occurring.[1]

The risk impact can be estimated based on the complexity of exploitation conditions (representing the likelihood) and the severity of exploitation results.

		Complexity of exploitation conditions		
		Simple	Moderate	Complex
Severity of exploitation results	Major	Critical	High	Medium
	Moderate	High	Medium	Low
	Minor	Medium	Low	Low

[1] NIST FIPS PUB 200: Minimum Security Requirements for Federal Information and Information Systems. Gaithersburg, MD: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

7. CONTACT

Person responsible for providing explanations:

Damian Rusinek
e-mail: Damian.Rusinek@securing.pl
tel.: +48 12 425 25 75

securing
http://www.securing.pl
e-mail: info@securing.pl
Kalwaryjska 65/6
30-504 Kraków
tel./fax.: +48 (12) 425 25