u2u network Smart Contract

SMART CONTRACT AUDIT REPORT

November 2024

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by u2u network to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	Informational	Low	Medium	High
High	Informational	Medium	High	Critical
Medium	Informational	Low	Medium	High
Low	Informational	Low	Low	Medium

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment

and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks

Category	Assessment Item
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: u2u network

Audit Time: November 8nd, 2024 – November 9th, 2024

Language: Solidity

File Name	HASH
Incentive-campaign-smc	https://github.com/unicornultrafoundation/incentive-campaign-smc/releases/tag/v0.0.3

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	1	
Low	3	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	Medium	Rewards may continue to accumulate after endTime	Fixed	Confirmed
NVE-002	Low	In extreme cases, the pledged funds will overflow	Fixed	Confirmed
NVE-003	Low	There may be a situation where the staking reward funds are received but the contract reward funds are insufficient	Fixed	Confirmed
NVE-004	Low	Privileged role risks	Acknowledged	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Rewards may continue to accumulate after endTime

ID:	NVE-001	Location:	staking/IncentivePool.sol
Severity:	Medium	Category:	Business Issues
Likelihood:	Low	Impact:	High

Description:

In the _pendingRewards method, the user's reward is calculated based on the time difference between latestHarvest and the current time block.timestamp.

uint256 timeRewards = block.timestamp - latestHarvest;

endTime is not checked here, so even after endTime, _pendingRewards will still calculate the time from latestHarvest to the current time as the reward time, causing users to continue to accumulate rewards after the staking period ends.

```
function _pendingRewards(address _user) internal view returns (uint256) {
    unchecked {
        (uint256 totalStaked, uint256 latestHarvest, ) = _getUserInfo(
            _user
        );
        if (block.timestamp < startTime) return 0;
        if (latestHarvest < startTime) {
            latestHarvest = startTime;
        }
        if (totalStaked == 0) return 0;
        uint256 timeRewards = block.timestamp - latestHarvest;
        return timeRewards.mul(totalStaked).mul(_rewardsRatePerSecond());
    }
}
```

Recommendations:

Exvul Web3 Security recommends adding a check for endTime in the _pendingRewards and _harvest methods to ensure that rewards do not continue to accumulate after endTime.

Result: Confirmed

Fix Result: Fixed

The client has added a checksum to avoid this risk.

Fixed version:

https://github.com/unicornultrafoundation/incentive-campaign-smc/releases/tag/v0.0.4.

```
280     if (latestHarvest > endTime) {
281         latestHarvest = endTime;
282     }
283     if (0 == totalStaked) return 0;
284     uint256 checkPointTime = block.timestamp < endTime
285         ? block.timestamp
286         : endTime;
287     uint256 timeRewards = checkPointTime - latestHarvest;
```

3.2 In extreme cases, the pledged funds will overflow

ID:	NVE-002	Location:	staking/IncentivePool.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

The flow of the contract's _stake method is as follows:

Call _harvest() to update the user's staking status and rewards.

Use unchecked to perform accumulation operations users_[_user].totalStaked += _amount; and totalPoolStaked += _amount;.

Perform upper limit checks on users_[_user].totalStaked and totalPoolStaked respectively.

Because unchecked skips the overflow check, in extreme cases, when the staking amount _amount is superimposed to or close to the maximum value of uint256, an overflow may occur. After the overflow, the value of the variable will return to a lower range and may be mistakenly considered to be within the set upper limit, thereby bypassing two conditional checks:

users_[_user].totalStaked <= MAX_STAKE_AMOUNT: Check whether the user's staking amount exceeds the personal upper limit.

totalPoolStaked <= MAX_USDT_POOL_CAP: Check whether the staking amount of the entire pool exceeds the pool upper limit.

The overflow bypass mechanism may cause the following problems:

Bypassing the individual staking limit: If users_[_user].totalStaked overflows to a lower value (e.g. close to 0), the user can perform the staking operation again, breaking the MAX_STAKE_AMOUNT limit.

Bypassing the total pool limit: The same overflow mechanism can make totalPoolStaked lower than MAX_USDT_POOL_CAP, thereby allowing further staking, causing the total amount of funds in the pool to get out of control.

```
182     function _stake(uint256 _amount) internal {
183         unchecked {
184             _harvest();
185             address _user = msg.sender;
186             users_[_user].totalStaked += _amount;
187             require(
188                 users_[_user].totalStaked <= MAX_STAKE_AMOUNT,
189                 "staked amount over"
190             );
191             totalPoolStaked += _amount;
192             require(totalPoolStaked <= MAX_USDT_POOL_CAP, "maximum pool cap");
193 
194             // Send USDT to staking pool
195             TransferHelper.safeTransferFrom(
196                 pUSDT,
197                 _user,
198                 address(this),
199                 _amount
200             );
201             emit Stake(_user, _amount);
202         }
203     }
```

Recommendations:

Exvul Web3 Security recommends removing the unchecked block or adding checks to prevent overflow.

Result: Confirmed

Fix Result: Fixed

The client has unchecked.

Fixed version:

https://github.com/unicornultrafoundation/incentive-campaign-smc/releases/tag/v0.0.4.

```
193     function _stake(uint256 _amount) internal {
194         _updateDebt();
195         address _user = msg.sender;
196         users_[_user].totalStaked += _amount;
197         require(
198             users_[_user].totalStaked <= MAX_STAKE_AMOUNT,
199             "staked amount over"
200         );
201         totalPoolStaked += _amount;
202         require(totalPoolStaked <= MAX_USDT_POOL_CAP, "maximum pool cap");
203 
204         // Send USDT to staking pool
205         TransferHelper.safeTransferFrom(pUSDT, _user, address(this), _amount);
206         uint256 _rewards = _estimateRewards(_amount);
207         totalAmountOwedToUsers += _rewards;
208         emit Stake(_user, _amount);
209     }
```

3.3 There may be a situation where the staking reward funds are received but the contract reward funds are insufficient

ID:	NVE-003	Location:	staking/IncentivePool.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

In the _harvest method, the user's reward funds are transferred to the user through TransferHelper.safeTransferNative. However, if the contract balance is not enough to pay the user's reward, this transfer operation will fail and trigger a fallback. Since the contract does not check the balance in the reward pool in advance, this may lead to the risk that the user cannot withdraw the reward normally.

```
205     function _harvest() internal {
206         unchecked {
207             address _user = msg.sender;
208             uint256 _u2uRewards = _pendingRewards(_user);
209             users_[_user].latestHarvest = block.timestamp;
210             if (_u2uRewards > 0) {
211                 users_[_user].totalClaimed += _u2uRewards;
212                 // Handle send rewards
213                 TransferHelper.safeTransferNative(_user, _u2uRewards);
214                 emit Harvest(_user, _u2uRewards);
215             }
216         }
217     }
```

Recommendations:

Exvul Web3 Security recommends checking contract balances before transferring money.

Result: Confirmed

Fix Result: Fixed

The client has added a checksum to avoid this risk.

Fixed version:

https://github.com/unicornultrafoundation/incentive-campaign-smc/releases/tag/v0.0.4.

```
231     function _harvest() internal {
232         address _user = msg.sender;
233         uint256 _u2uRewards = _pendingRewards(_user);
234         users_[_user].latestHarvest = block.timestamp;
235         if (_u2uRewards > 0) {
236             require(
237                 address(this).balance >= _u2uRewards,
238                 "Pool rewards insufficient"
239             );
240             users_[_user].totalClaimed += _u2uRewards;
241             users_[_user].debt = 0;
242             // Handle send rewards
243             TransferHelper.safeTransferNative(_user, _u2uRewards);
244             totalAmountOwedToUsers = totalAmountOwedToUsers > _u2uRewards
245             emit Harvest(_user, _u2uRewards);
246         }
247     }
```

3.4 Privileged role risks

ID:	NVE-004	Location:	staking/IncentivePool.sol aidrop/AirdropPool.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

Privileged role risks in the AirdropPool contract

In the AirdropPool contract, there are several privileged roles and corresponding permissions:

DEFAULT_ADMIN_ROLE: Set by the deployer of the contract, it can manage the permissions of all other roles. If the private key of this role is leaked, the malicious party will be able to change or add other roles and gain full control of the contract.

AIRDROP_ADMIN: Has the right to call the sendAir function, that is, control the airdrop operation. This role determines who can receive the airdrop reward. If the private key of the AIRDROP_ADMIN role is leaked, the attacker can control the issuance of airdrops and even issue airdrops to any account.

POOL_SIGNER: This role verifies the signature through _verify to ensure that only specific signers can authorize the sendAir operation. If the private key of this role is leaked, the attacker can forge a signature and perform an airdrop through the sendAir function.

In addition, an emergencyWithdrawU2U function is provided in the contract, which allows the onlyMasterAdmin (i.e. DEFAULT_ADMIN_ROLE) role to directly withdraw funds from the contract. If the private key of this role is lost or stolen, the attacker can quickly withdraw all the funds in the contract.

Risks of privileged roles in the IncentivePool contract

The IncentivePool contract contains the following privileged roles:

DEFAULT_ADMIN_ROLE: Set by the deployer of the contract, it has the highest authority and can manage the permissions of all other roles. If the private key of this role is leaked, the attacker can reset the contract parameters, assign new roles, and even control all the logic of the entire contract.

POOL_SIGNER: Used to verify the identity of the pledge signer to ensure that the request to call the stake method comes from a legitimate signer. If the private key of POOL_SIGNER is leaked, the

attacker can forge a signature and bypass the verification process, thereby performing malicious operations through the stake function.

onlyMasterAdmin call permission: The emergencyWithdrawU2U function in the contract allows the DEFAULT_ADMIN_ROLE to withdraw funds from the contract. If the administrator's private key is leaked, an attacker can use this permission to withdraw all funds in the contract.

Claimable Time and Ignore Signer settings: only the MasterAdmin role can call the setClaimableTime and setIgnoreSigner functions to set the time for reward withdrawal and whether to ignore signer verification. If these privileges are abused, attackers can bypass signature verification or arbitrarily set the time for reward issuance, affecting the normal reward acquisition of users.

Recommendations:

Exvul Web3 Security recommends using multi-signature wallets as privileged role addresses, or introducing a time lock mechanism to ensure that important operations (such as fund withdrawals, role assignments) require a delay period, allowing the community or administrators to discover and prevent malicious operations during the delay period.

Result: Confirmed

Fix Result: Acknowledged

The u2u Network team stated that the AirdropPool contract is for internal use only.

At present, the client has updated the logic of the IncentivePool contract to transfer funds, so that the calculated rewards cannot be transferred out. This is safe for the user's rewards in actual situations, but there is an unexpected situation. If some users do not receive the rewards, then these funds will lock the users in the contract.

Acknowledged version:

https://github.com/unicornultrafoundation/incentive-campaign-smc/releases/tag/v0.0.4.

```
308     function emergencyWithdrawU2U(
309         address _to,
310         uint256 _amount
311     ) external onlyMasterAdmin {
312         require(
313             _amount <= address(this).balance - totalAmountOwedToUsers,
314             "invalid amount"
315         );
316         TransferHelper.safeTransferNative(_to, _amount);
317     }
318 }
```

4. CONCLUSION

In this audit, we thoroughly analyzed **u2u network** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be Passed. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

www.exvul.com

contact@exvul.com

@EXVULSEC

github.com/EXVUL-Sec

ExVul