BulbaSwap Smart Contract

SMART CONTRACT AUDIT REPORT

October 2024

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by BulbaSwap 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
	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

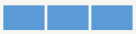
Project Name: BulbaSwap

Audit Time: October 10, 2024 – October 12, 2024

Language: Solidity

File Name	HASH
BulbaSwap	https://github.com/BulbaSwap/uniswap-v2/commit/cb7330d3dc2d71d822429b9d394cea57d1e99436

2.2 Summary

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

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	Low	Sandwich Attack	Fixed	Confirmed
NVE-002	Low	Low Liquidity Pool Price Manipulation	Fixed	Confirmed
NVE-003	Low	Impermanent Loss Risk	Fixed	Confirmed
NVE-004	Informational	Approval Vulnerability	Fixed	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Sandwich Attack

ID:	NVE-001	**Location:**	contracts/v2-periphery/contracts/UniswapV2Router01.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

In the transaction exchange, the transaction price is dynamically calculated based on the reserve of the two tokens in the pool through the constant product formula x * y = k. On blockchain networks such as Ethereum, the execution order of transactions depends on the gas fee. Sandwich attack is an attack mode that uses this mechanism. The attacker can manipulate the price by inserting his own transactions, causing the target user to suffer losses.

The specific steps of the sandwich attack are:

Pre-transaction: The attacker first issues a token purchase transaction, and the ratio of reserveIn and reserveOut changes, causing amountOut to decrease and push up the price.

User's transaction execution: The user's transaction price has risen, resulting in a higher cost to complete the transaction, resulting in high slippage.

Post-transaction: The attacker sells the token again, restores the price and profits from it.

Recommendations:

Exvul Web3 Security recommends that users should set a reasonable slippage tolerance when initiating transactions to avoid transactions being executed at unreasonable prices due to price fluctuations. Introducing the time-weighted average price can reduce the possibility of price manipulation in a short period of time.

Result: Confirmed

Fix Result: Fixed

3.2 Low Liquidity Pool Price Manipulation

ID:	NVE-002	Location:	contracts/v2-core/contracts/UniswapV2Pair.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

When users trade, the price is determined by the two asset reserves in the pool. The constant product formula x * y = k controls the balance between two assets, where x and y represent the reserves of the two assets respectively, and k is constant. This means that reserve changes instantly change the price in the pool every time a trade occurs.

When pool liquidity is low, the values of reserves x and y are very small, so even a small transaction can significantly change the ratio of the two tokens, causing wild price swings. An attacker can take advantage of this and manipulate the price in the pool through a small transaction, causing a deviation from the market price, and profit through arbitrage in other more liquid markets.

Attack Steps:

1. The attacker chooses a low liquidity pool to trade.

2. Through small transactions, the reserves in the pool are manipulated to cause a large change in the token ratio (i.e., price), causing the price in the pool to deviate significantly from the market price.

3. The attacker uses this price deviation to arbitrage in other high-liquidity markets.

4. After the transaction is completed, the price of Uniswap is restored, but users or liquidity providers may have suffered losses.

```
// this low-level function should be called from a contract which performs important safety checks
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
    require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');

    uint balance0;
    uint balance1;
    { // scope for _token{0,1}, avoids stack too deep errors
    address _token0 = token0;
    address _token1 = token1;
    require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
    if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
    if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
    if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
    balance0 = IERC20(_token0).balanceOf(address(this));
    balance1 = IERC20(_token1).balanceOf(address(this));
    }
    uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
    uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
    require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
    { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
    uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(35));
    uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(35));
    require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(10000**2), 'UniswapV2: K');
    }

    _update(balance0, balance1, _reserve0, _reserve1);
    emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
}
```

Recommendations:

Exvul Web3 Security recommends introducing external price oracles (such as Chainlink) to compare with Uniswap prices; the protocol front end should issue low liquidity warnings to users, reminding users that they may face high slippage and price manipulation risks when trading in pools with low liquidity.

Result: Confirmed

Fix Result: Fixed

3.3 Impermanent Loss Risk

ID:	NVE-003	Location:	contracts/v2-core/contracts/UniswapV2Pair.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

When the relative price of two tokens in the pool fluctuates, liquidity providers may suffer impermanent loss, especially in a highly volatile market environment. The greater the price change, the more severe the impermanent loss faced by liquidity providers. In the project, the core mechanism for impermanent loss is in the swap function of the UniswapV2Pair.sol contract. Through the token swap mechanism of this function, the token reserves x and y in the pool change with the transaction. AMM uses the constant product formula x * y = k, where x and y represent the reserves of the two tokens in the pool, respectively, and k is a constant value. When users trade, the values of x and y will change, which in turn affects the price. If the price fluctuates greatly, the assets obtained by liquidity providers when extracting liquidity may be lower than the original asset value.

```
// this low-level function should be called from a contract which performs important safety checks
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
    require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');

    uint balance0;
    uint balance1;
    { // scope for _token{0,1}, avoids stack too deep errors
    address _token0 = token0;
    address _token1 = token1;
    require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
    if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
    if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
    if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
    balance0 = IERC20(_token0).balanceOf(address(this));
    balance1 = IERC20(_token1).balanceOf(address(this));
    }
    uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
    uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
    require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
    { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
    uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(35));
    uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(35));
    require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(10000**2), 'UniswapV2: K');
    }

    _update(balance0, balance1, _reserve0, _reserve1);
    emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
}
```

Recommendations:

Exvul Web3 Security recommends that liquidity providers can choose relatively stable trading pairs (such as stablecoin pairs) to reduce the risk of impermanent loss.

Result: Confirmed

Fix Result: Fixed

3.4 Approval Vulnerability

ID:	NVE-004	Location:	contracts/v2-periphery/contracts/UniswapV2Router02.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Low	Impact:	Informational

Description:

In the ERC-20 standard, token authorization is performed through the approve function. When users interact with the platform, they usually call the approve function to authorize a smart contract to withdraw tokens from the user's account and complete the transaction. In order to reduce the tedious operation of authorization for each transaction, users often choose "unlimited approval" (that is, setting the authorization amount to uint(-1)), which means that the smart contract can use all tokens in the user's account at any time.

Specific Attack Method:

The user authorized the contract to use their tokens and selected unlimited approval (uint(-1)).

If the attacker finds a vulnerability in the contract, or the contract is replaced with a malicious version, the attacker can call the function in the contract to transfer the tokens authorized by the user.

Because the user chose unlimited authorization, the attacker can withdraw all tokens in the user's account without restriction, causing asset loss.

Recommendations:

Exvul Web3 Security recommends that users do not use unlimited authorization and revoke authorization in a timely manner. When guiding users to authorize tokens, the platform can warn users not to use unlimited approval and recommend setting a reasonable authorization limit.

Result: Confirmed

Fix Result: Fixed

4. CONCLUSION

In this audit, we thoroughly analyzed **BulbaSwap** 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