LetsPumpLaucher Smart Contract

SMART CONTRACT AUDIT REPORT

August 2024

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by LetsPumpLaucher 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: LetsPumpLaucher

Audit Time: August 1nd, 2024 – August 6th, 2024

Language: Solidity

File Name	Link
LetsPumpLaucher	https://scan.coredao.org/address/0x66EB01FeE5992029d43FadabA6AeD6806f075326#code

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	1	
Low	4	
Informational	1	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	Medium	Centralized role	Fixed	Confirmed
NVE-002	Low	The use of SafeMath library contracts leads to an increase in Gas	Fixed	Confirmed
NVE-003	Low	Calculation accuracy issues	Fixed	Confirmed
NVE-004	Low	The mintToken() method fee can be bypassed	Fixed	Confirmed
NVE-005	Low	Status updates may not be triggered	Fixed	Confirmed
NVE-006	Informational	The returned pageSize value is inaccurate	Fixed	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Centralized role

ID:	NVE-001	Location:	LetsPumpLaucher.sol
Severity:	Medium	Category:	Privileged role
Likelihood:	Low	Impact:	High

Description:

The current owner privilege role of the contract is EOA (0x104A49EBCdde29384A54d8856821D85b913D467a). If the EOA is maliciously manipulated, it may bring the following security risks:

Transfer all token assets：

The contract has a clearTarget method that allows the owner to transfer all tokens in the contract. The owner can call this function to transfer all specified ERC20 tokens in the contract to any address. If the owner is maliciously manipulated, funds may be stolen.

```
function clearTarget(address targetContract, address recipient) public onlyOwner {
    uint balance = IERC20(targetContract).balanceOf(address(this));
    if (balance > 0) {
        IERC20(targetContract).transfer(recipient, balance);
    }
}
```

Figure 3.1.1 clearTarget() function

Transfer the ETH balance in the contract：

There is a clear method in the contract that allows the owner to transfer the ETH balance in the contract. The owner can call this function to transfer all the ETH balance in the contract to any address. Similarly, if the owner is maliciously manipulated, these ETH may be stolen.

```
function clear(address recipient) public onlyOwner {
    if (address(this).balance > 0) {
        payable(recipient).transfer(address(this).balance);
    }
}
```

Figure 3.1.2 clear () function

Set important contract parameters：

There are multiple methods in the contract that allow the owner to set key parameters, such as:

setCreateFee(uint value): Set the fee for creating a token.

setLaunchFee(uint8 value): Set the launch fee.

setFeeRecipient(address target): Set the fee recipient address.

setWCORE(address target): Set the core token address.

setMinLimitAmount(uint value): Set the minimum limit amount.

If the owner is maliciously manipulated, these parameters may be set to unreasonable values, causing the normal operation of the contract to be affected. For example, a malicious manipulator can set feeRecipient to his own address, and subsequent fees will be transferred to this address.

```
function setCreateFee(uint value) public onlyOwner {
    _createFee = value;
}
function setLaunchFee(uint8 value) public onlyOwner {
    _launchFee = value;
}
function setFeeRecipient(address target) public onlyOwner {
    _feeRecipient = target;
}
function setWCORE(address target) public onlyOwner {
    _WCORE = target;
}
function setMinLimitAmount(uint value) public onlyOwner {
    _minLimitAmount = value;
    _stepAmount = value / 100000;
}
```

Figure 3.1.3 set function

Fundraising Operations：

There is a safeBack method in the contract that allows the owner to control the fundraising status. The safeBack(address target) method can be used to mark the token issuance as failed.

```
function safeBack(address target) public onlyOwner IsValidToken(target) {
    Round memory round = _rounds[target];
    require(round.status == Status.Launch, "E1");
    require(block.timestamp - round.mintEnd > 1 days, "E2");
    _rounds[target].status = Status.Failed;
}
```

Figure 3.1.4 safeBack() function

Recommendations:

Exvul Web3 Security recommends that when transferring funds, it is necessary to exclude the funds involved in the fund raising in the contract to avoid fund losses;

Multi-signature wallets, the owner authority is managed by a multi-signature wallet to reduce the risk of single point failure;

Authority division, assign different permissions to different roles instead of concentrating them on the owner role, reducing the impact of malicious manipulation of a single role..

Result: Confirmed

Fix Result: Fixed

3.2 The use of SafeMath library contracts leads to an increase in Gas

ID:	NVE-002	**Location:**	LetsPumpLaucher.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

The current contract is solidity ^0.8.0 version, and uses the SafeMath library. Since in Solidity 0.8 and above, the compiler already has built-in overflow checking, this means that the SafeMath library is actually redundant if these versions are used. So using SafeMath in Solidity 0.8+ does not add additional security risks, but it will increase unnecessary gas costs because each mathematical operation will be checked twice: once automatically by the Solidity compiler, and once by the SafeMath function.

```sol
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}
```

Figure 3.2.1 SafeMath library

Recommendations:

Exvul Web3 Security recommends removing the SafeMath library.

Result: Confirmed

Fix Result: Fixed

3.3 Calculation accuracy issues

ID:	NVE-003	Location:	LetsPumpLaucher.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

In the mintToken method, the calculation logic of uint payAmount = msg.value / _stepAmount * _stepAmount; is to adjust the msg.value (in Wei) paid by the user to a value that conforms to the precision of _stepAmount. Specifically, it rounds the payment amount down to the nearest multiple of _stepAmount.

There is a problem of precision loss here because it rounds msg.value down to the nearest multiple of _stepAmount through integer division and multiplication. For example, if _stepAmount is 1000 Wei and the msg.value paid by the user is 1500 Wei, then the result of payAmount will be 1000 Wei, 500 Wei is discarded. Any part of msg.value that cannot be divided by _stepAmount will be discarded. For users, this means that part of the amount they pay will not be included in payAmount, thereby reducing the number of tokens they actually receive.

```
function mintToken(address target) public payable isHuman nonReentrant IsValidToken(target) {
    address sender = _msgSender();
    Round memory round = _rounds[target];
    require(round.status == Status.Mint, "E1");

    uint payAmount = msg.value / _stepAmount * _stepAmount;
    require(payAmount > 0, "E4");

    require(payAmount >= round.minPay && payAmount <= round.maxPay, "E2");
    require(block.timestamp <= round.mintEnd, "E3");
    require(payAmount + round.payTotal <= round.payLimit, "E5");

    uint mintAmount = payAmount * round.mintPrice / 1e18;
    require(mintAmount > 0, "E6");
```

Figure 3.3.1 SafeMath library

Recommendations:

Exvul Web3 Security recommends optimizing the computing and fee collection model.

Result: Confirmed

Fix Result: Fixed

3.4 The mintToken() method fee can be bypassed

ID:	NVE-004	**Location:**	LetsPumpLaucher.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

The mintToken method charges a handling fee, which is calculated by msg.value - payAmount. msg.value is the total amount paid by the user, and payAmount is the value rounded down to the nearest multiple of _stepAmount. The difference here is the handling fee. This difference, msg.value - payAmount, will be transferred to the _feeRecipient address.

The problem here is that the handling fee can be bypassed by precisely controlling msg.value. If the msg.value paid by the user happens to be a multiple of _stepAmount, msg.value - payAmount will be equal to zero, so no handling fee will be charged. For example, if msg.value = 10000 Wei, and _stepAmount = 1000 Wei, then payAmount is 10000 Wei, and the handling fee is zero.

```
function mintToken(address target) public payable isHuman nonReentrant IsValidToken(target) {
    address sender = _msgSender();
    Round memory round = _rounds[target];
    require(round.status == Status.Mint, "E1");

    uint payAmount = msg.value / _stepAmount * _stepAmount;
    require(payAmount > 0, "E4");

    require(payAmount >= round.minPay && payAmount <= round.maxPay, "E2");
    require(block.timestamp <= round.mintEnd, "E3");
    require(payAmount + round.payTotal <= round.payLimit, "E5");

    uint mintAmount = payAmount * round.mintPrice / 1e18;
    require(mintAmount > 0, "E6");

    MintDetail memory detail;
    detail.target = sender;
    detail.time = block.timestamp;
    detail.payAmount = payAmount;
    detail.getAmount = mintAmount;

    _mints[sender][round.token].details.push(detail);
    _mints[sender][round.token].totalPay += payAmount;
    _mints[sender][round.token].totalReceive += mintAmount;

    _roundMints[round.token].push(detail);

    _rounds[target].payTotal += payAmount;
    _rounds[target].mintTotal += mintAmount;


    if (_rounds[target].payTotal == _rounds[target].payLimit) {
        _rounds[target].status = Status.Launch;
        emit OnMintDone(round.token);
    }

    if (msg.value - payAmount > 0) {
        payable(_feeRecipient).transfer(msg.value - payAmount);
    }
```

Figure 3.4.1 SafeMath library

Recommendations:

Exvul Web3 Security recommends that if you need to charge a fee, it is recommended to use a fixed ratio to charge the fee. You can charge a fixed ratio of the fee directly from msg.value instead of based on rounding.

Result: Confirmed

Fix Result: Fixed

3.5 Status updates may not be triggered

ID:	NVE-005	Location:	LetsPumpLaucher.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

In the mintToken method, the Status.Launch state is modified by the judgment of _rounds[target].payTotal == _rounds[target].payLimit. The payAmount is calculated. The current value of _stepAmount in the contract is 10 ** 17 / 100000. The problem here is: if the _rounds[target].payTotal added by the user is infinitely close to _rounds[target].payLimit, but has not reached _rounds[target].payLimit, but the payAmount value cannot be calculated to a very small value, making _rounds[target].payTotal == _rounds[target].payLimit unsatisfactory.

Assumption:

_rounds[target].payLimit = 1000000000000000 Wei (i.e. 0.001 ETH)

Currently _rounds[target].payTotal = 9999999999999999 Wei (i.e. 0.0099999999999999999 ETH)

_stepAmount = 10 ** 12 Wei

At this point, _rounds[target].payTotal is 1 Wei away from payLimit, but because the value of _stepAmount is 10 ** 12 Wei, the user cannot pay 1 Wei, and the minimum payment unit is 1e12 Wei.

Since the user cannot pay a small enough amount to make _rounds[target].payTotal reach _rounds[target].payLimit, the state of the contract may not be updated to Status.Launch, resulting in the inability of fundraising to enter the next step.

```
function mintToken(address target) public payable isHuman nonReentrant IsValidToken(target) {
    address sender = _msgSender();
    Round memory round = _rounds[target];
    require(round.status == Status.Mint, "E1");

    uint payAmount = msg.value / _stepAmount * _stepAmount;
    require(payAmount > 0, "E4");

    require(payAmount >= round.minPay && payAmount <= round.maxPay, "E2");
    require(block.timestamp <= round.mintEnd, "E3");
    require(payAmount + round.payTotal <= round.payLimit, "E5");

    uint mintAmount = payAmount * round.mintPrice / 1e18;
    require(mintAmount > 0, "E6");

    MintDetail memory detail;
    detail.target = sender;
    detail.time = block.timestamp;
    detail.payAmount = payAmount;
    detail.getAmount = mintAmount;

    _mints[sender][round.token].details.push(detail);
    _mints[sender][round.token].totalPay += payAmount;
    _mints[sender][round.token].totalReceive += mintAmount;

    _roundMints[round.token].push(detail);

    _rounds[target].payTotal += payAmount;
    _rounds[target].mintTotal += mintAmount;


    if (_rounds[target].payTotal == _rounds[target].payLimit) {
        _rounds[target].status = Status.Launch;
        emit OnMintDone(round.token);
    }

    if (msg.value - payAmount > 0) {
        payable(_feeRecipient).transfer(msg.value - payAmount);
    }

    emit OnMint(round.token, sender, payAmount, mintAmount);
}
```

Figure 3.5.1 SafeMath library

Recommendations:

Exvul Web3 Security recommends adjusting the precision of _stepAmount if the contract's status may not be updated to Status.Launch and requires dynamic adjustment of privileged roles.

Result: Confirmed

Fix Result: Fixed

3.6 The returned pageSize value is inaccurate

ID:	NVE-006	Location:	LetsPumpLaucher.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Low	Impact:	Informational

Description:

The roundMintList method is used to obtain the specific fundraising details of each token. If the pageSize value here is greater than 100, 10 will be displayed. According to the logic, the caller wants to display or obtain more, but only 10 will be displayed. It is recommended to change pageSize > 100 here to display pageSize as 100.

```
function roundMintList(address token, uint pageNo, uint pageSize) public view returns(MintDetail[] memory list, uint total) {
    total = _roundMints[token].length;
    if (total > 0) {
        pageSize = (pageSize > 100 || pageSize == 0) ? 10: pageSize;
        uint pageIndex = (pageNo - 1) * pageSize;
        uint begin = total > pageIndex ? total - pageIndex: 0;
        if (begin > 0) {
            uint end = begin > pageSize ? begin - pageSize: 0;
            list = new MintDetail[](begin < pageSize ? begin: pageSize);
            for (uint i = begin; i > end; i--) {
                list[begin - i] = _roundMints[token][i - 1];
            }
        }
    }
}
```

Figure 3.6.1 setTransactionFee() function

Recommendations:

Exvul Web3 Security recommends changing pageSize > 100 here to display pageSize as 100.

Result: Confirmed

Fix Result: Fixed

4. CONCLUSION

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