



Audit report of Commodity Vault

Prepared By: - Kishan Patel
Prepared On: - 27/02/2024.

Prepared for: Anthony K. miller

Table of contents

- 1. Disclaimer**
- 2. Introduction**
- 3. Project information**
- 4. List of attacks checked**
- 5. Severity Definitions**
- 6. Good things in code**
- 7. Critical vulnerabilities in code**
- 8. Medium vulnerabilities in code**
- 9. Low vulnerabilities in code**
- 10. Summary**

THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.

THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

1. Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). Because the total numbers of test cases are unlimited, the audit makes no statements or warranties on the security of the code.

It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

2. Introduction

Kishan Patel (Consultant) was contacted by Anthony k. miller. (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between 27/02/2024 – 28/02/2024.

The project has 1 file. It contains approx 300 lines of Solidity code. All the functions and state variables are well commented on using the natspec documentation, but that does not create any vulnerability.

3. Project information

Token Name	Commodity Vault
Token Symbol	COVO
Platform	Polygon
Order Started Date	27/02/2024
Order Completed Date	28/02/2024

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BNB to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference

5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution

6. Good things in code

- **Good required condition in functions:-**

- Here smart contract is checking that newOwner address is valid and proper.

```
167     function transferOwnership(address newOwner) public virtual onlyOwner
168     {
169         if (newOwner == address(0)) {
170             revert OwnableInvalidOwner(address(0));
171         }
172         _transferOwnership(newOwner);
173     }
174 }
```

- Here smart contract is checking that from and to addresses are valid and proper.

```
455     function _transfer(address from, address to, uint256 value) internal
456     {
457         if (from == address(0)) {
458             revert ERC20InvalidSender(address(0));
459         }
460         if (to == address(0)) {
461             revert ERC20InvalidReceiver(address(0));
462         }
463     }
464 }
```

- Here smart contract is checking that account address is valid and proper.

```
509     */
510     function _mint(address account, uint256 value) internal {
511         if (account == address(0)) {
512             revert ERC20InvalidReceiver(address(0));
513         }
514         _update(address(0), account, value);
515     }
516 }
```

```
525     function _burn(address account, uint256 value) internal {
526         if (account == address(0)) {
527             revert ERC20InvalidSender(address(0));
528         }
529         _update(account, address(0), value);
530     }
531 }
```


- Here smart contract is checking that owner and spender addresses are valid and proper.

```

568     function _approve(address owner, address spender, uint256 value, bool
569         if (owner == address(0)) {
570             revert ERC20InvalidApprover(address(0));
571         }
572         if (spender == address(0)) {
573             revert ERC20InvalidSpender(address(0));
574         }

```

- Here smart contract is checking that transfer to owner is successfully done or not.

```

640     function claimStuckedERC20(address token, uint256 amount) external
641         (bool success, bytes memory data) = token.call(
642             abi.encodeWithSelector(0xa9059cbb, owner(), amount)
643         );
644         require(
645             success && (data.length == 0 || abi.decode(data, (bool))),
646             "ERC20: TOKEN_CLAIM_FAILED"
647         );

```

- Here smart contract is checking that msg.sender is minter or not, totalSupply + value is not bigger than MAX_SUPPLY.

```

653     function mint (address to, uint256 amount) external {
654         if(!isMinter[msg.sender]){revert OnlyAuthorizedMinterCanCallThis}
655         if(totalSupply() + amount > MAX_SUPPLY){revert MaxSupplyExceeded}
656         _mint(to, amount);

```

- Here smart contract is checking that from or to addresses is not frozen.

```

659     // override required by solidity
660     function _update (address from, address to, uint256 amount) internal
661         if(isFrozen[from] || isFrozen[to]){revert UserTokensAreFrozen};
662         super._update(from, to, amount);
663     }

```

7. Critical vulnerabilities in code

- No Critical vulnerabilities found

8. Medium vulnerabilities in code

- No Medium vulnerabilities found

9. Low vulnerabilities in code

9.1. Suggestions to add code validations:-

=> You have implemented required validation in contract.

=> There are some place where you can improve validation and security of your code.

=> These are all just suggestion it is not bug.

○ Function: - `_burn`

```
524     */
525     function _burn(address account, uint256 value) internal {
526         if (account == address(0)) {
527             revert ERC20InvalidSender(address(0));
528         }
529         _update(account, address(0), value);
530     }
```

- Here in `_burn` function smart contract can check that account address has sufficient balance to burn.

○ **Function: - _approve**

```
568     function _approve(address owner, address spender, uint256 value, bool approved) public {
569         if (owner == address(0)) {
570             revert ERC20InvalidApprover(address(0));
571         }
572         if (spender == address(0)) {
573             revert ERC20InvalidSpender(address(0));
574         }
575     }
```

- Here in _approve function smart contract can check that owner account has sufficient balance to give allowance to spender address.

10. Summary

- **Number of problems in the smart contract as per severity level**

Critical	Medium	Low
0	0	2

According to the assessment, the smart contract code is well secured. The code is written with all validation and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- **Good Point:** Code performance and quality are good. All kind of necessary validation added into smart contract and all validations are working as expected.
- **Suggestions:** Please try to implement suggested code validations.