Audit report of Rhodium

Prepared By: - Kishan Patel
Prepared On: - 07/03/2024.

Prepared for: Evens

Table of contents

THIS AUDIT REPOR WILL CONTAIN CONFIDENTIAL INFORMATION T 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 auditcan’t guarantee explicit security of the audited smart contracts.

2. Introduction

Kishan Patel (Consultant) was contacted by Evens Saint Clair. (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 07/03/2024 – 09/03/2024.

The project has 1 file. It contains approx 700 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	Rhodium
Token Symbol	RHO
Platform	Polygon
Order Started Date	07/03/2024
Order Completed Date	09/03/2024

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 tokenslose
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.

```
173     */
174     function transferOwnership(address newOwner) public virtual onlyOwne
175         if (newOwner == address(0)) {
176             revert OwnableInvalidOwner(address(0));
177         }
178         _transferOwnership(newOwner);
```

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

```
480     function _transfer(
481         address from,
482         address to,
483         uint256 value
484     ) internal {
485         if (from == address(0)) {
486             revert ERC20InvalidSender(address(0));
487         }
488         if (to == address(0)) {
489             revert ERC20InvalidReceiver(address(0));
490         }
```

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

```
542      */
543     function _mint(address account, uint256 value) internal {
544         if (account == address(0)) {
545             revert ERC20InvalidReceiver(address(0));
546         }
547         _update(address(0), account, value);
```

```
557      */
558     function _burn(address account, uint256 value) internal {
559         if (account == address(0)) {
560             revert ERC20InvalidSender(address(0));
561         }
```

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

```
604      */
605     function _approve(
606         address owner,
607         address spender,
608         uint256 value,
609         bool emitEvent
610     ) internal virtual {
611         if (owner == address(0)) {
612             revert ERC20InvalidApprover(address(0));
613         }
614         if (spender == address(0)) {
615             revert ERC20InvalidSpender(address(0));
616         }
```

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

```
721     function claimStuckedERC20(address token, uint256 amount)
722         external
723         onlyOwner
724     {
725         (bool success, bytes memory data) = token.call(
726             abi.encodeWithSelector(0xa9059cbb, owner(), amount)
727         );
728         require(
729             success && (data.length == 0 || abi.decode(data, (bool))),
730             "ERC20: TOKEN_CLAIM_FAILED"
731         );
```

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

```
737     function mint(address to, uint256 amount) external {
738         if (!isMinter[msg.sender]) {
739             revert OnlyAuthorizedMinterCanCallThis();
740         }
741         if (totalSupply() + amount > MAX_SUPPLY) {
742             revert MaxSupplyExceeded();
743         }
```

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

```
747     /// override required by solidity
748     function _update(
749         address from,
750         address to,
751         uint256 amount
752     ) internal override {
753         if (isFrozen[from] || isFrozen[to]) {
754             revert UserTokensAreFrozen();
755         }
```

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 incontract.
=> 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**

```
558     function _burn(address account, uint256 value) internal {
559         if (account == address(0)) {
560             revert ERC20InvalidSender(address(0));
561         }
562         _update(account, address(0), value);
563     }
```

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

- **Function: - _approve**

```
604     */
605     function _approve(
606         address owner,
607         address spender,
608         uint256 value,
609         bool emitEvent
610     ) internal virtual {
611         if (owner == address(0)) {
612             revert ERC20InvalidApprover(address(0));
613         }
614         if (spender == address(0)) {
615             revert ERC20InvalidSpender(address(0));
616         }
```

- 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 performingwell 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 excepted.

- **Suggestions:** Please try to implement suggested code validations.