Zettahash

Audit report of Zettahash

Prepared By: - Kishan Patel
Prepared On: - 04/09/2023

Prepared for: Zettahash.

Table of contents

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 Zettahash. (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 04/09/2023 – 06/09/2023.

The project has 2 files. It contains approx 1200 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	Zettahash
Token Symbol	ZH
Platform	Ethereum
Order Started Date	04/09/2023
Order Completed Date	06/09/2023

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

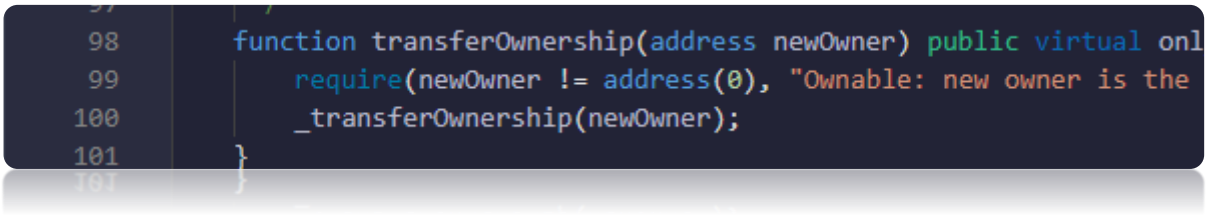
Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc.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

6. Good things in code

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

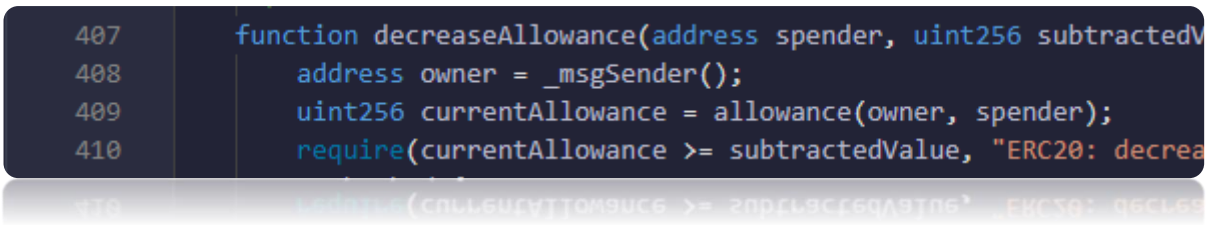
Filename: ZH.sol

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



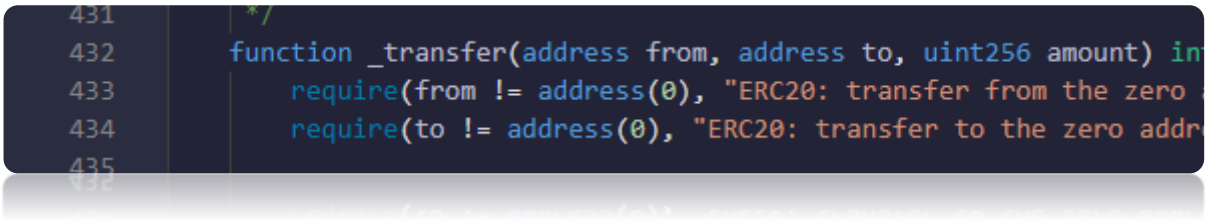
```
98  function transferOwnership(address newOwner) public virtual onl
99      require(newOwner != address(0), "Ownable: new owner is the
100     _transferOwnership(newOwner);
101 }
```

- Here smart contract is checking that currentAllowance is bigger or equal to subtractedValue.



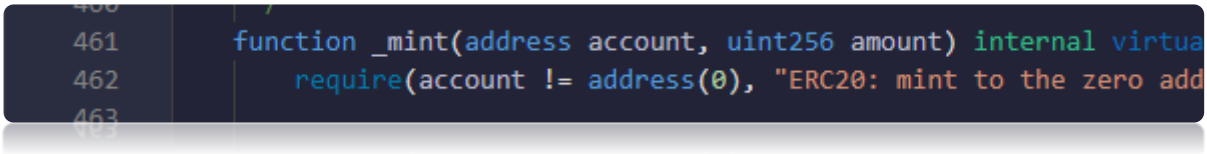
```
407 function decreaseAllowance(address spender, uint256 subtractedV
408     address owner = _msgSender();
409     uint256 currentAllowance = allowance(owner, spender);
410     require(currentAllowance >= subtractedValue, "ERC20: decrea
```

- Here smart contract is checking that from and to addresses are valid and proper, and balance of from address is bigger or equal to amount.



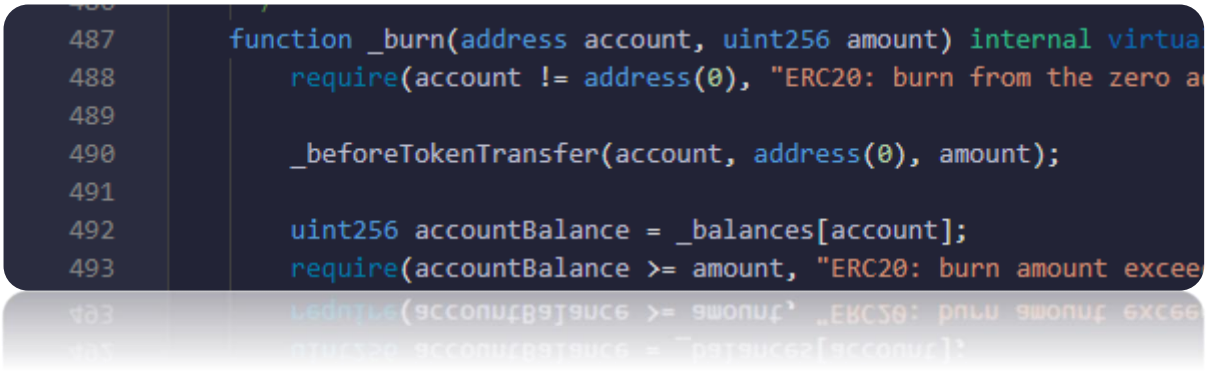
```
431  */
432 function _transfer(address from, address to, uint256 amount) in
433     require(from != address(0), "ERC20: transfer from the zero
434     require(to != address(0), "ERC20: transfer to the zero addr
435
```

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

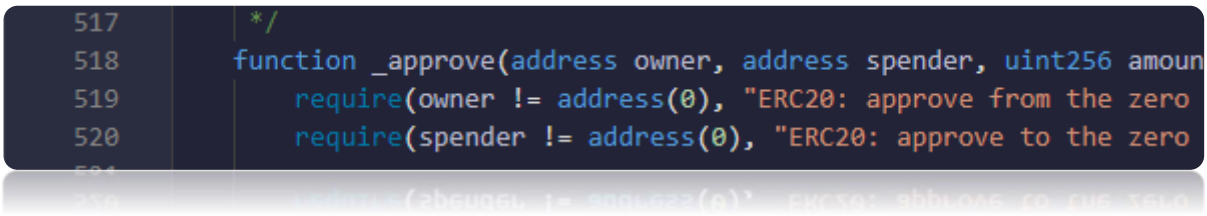


```
461 function _mint(address account, uint256 amount) internal virtua
462     require(account != address(0), "ERC20: mint to the zero add
463
```

- Here smart contract is checking that account address is valid and proper, balance of account address is bigger or equal to amount.

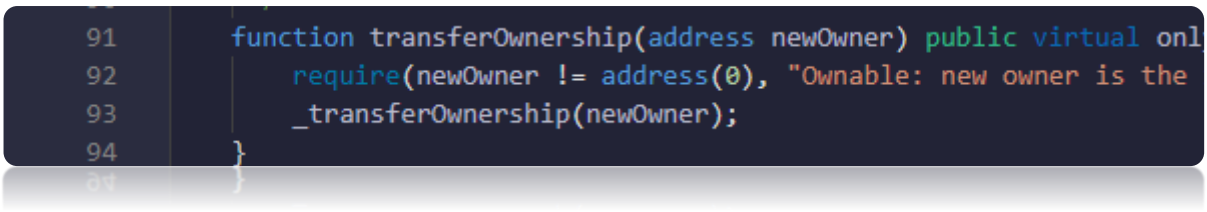


- Here smart contract is checking that owner and spender addresses are valid and proper.

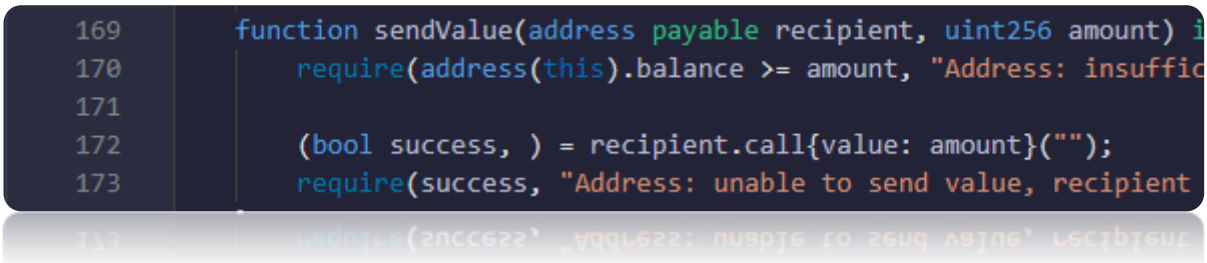


Filename: ZettahashHoldings.sol

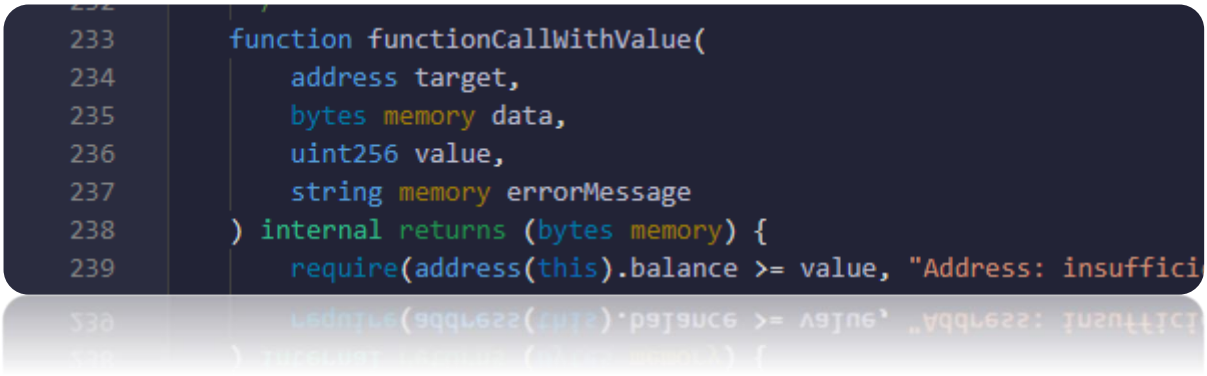
- Here smart contract is checking that newOwner address is valid and proper.



- Here smart contract is checking that balance of contract is bigger or equal to amount, transfer call to recipient is successfully done.

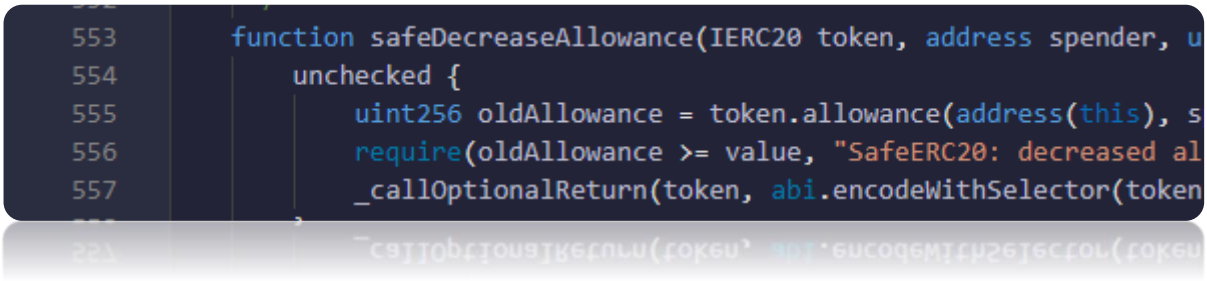


- Here smart contract is checking that balance of contract is bigger or equal to value.



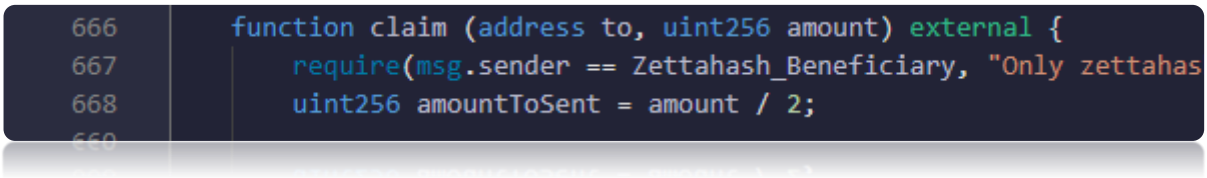
```
233     function functionCallWithValue(
234         address target,
235         bytes memory data,
236         uint256 value,
237         string memory errorMessage
238     ) internal returns (bytes memory) {
239         require(address(this).balance >= value, "Address: insuffici
```

- Here smart contract is checking that oldAllowance of contract is bigger or equal to value.



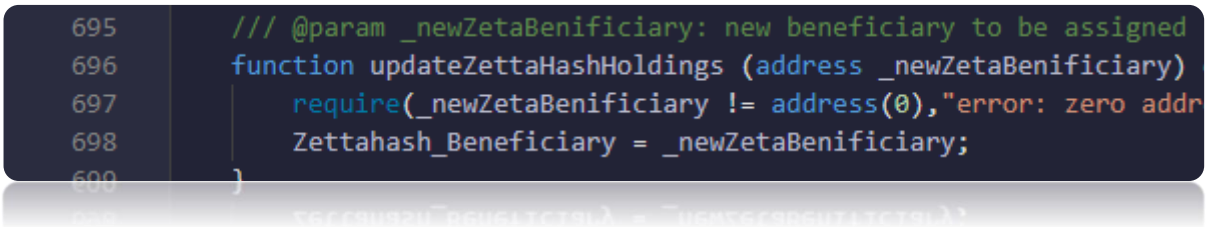
```
553     function safeDecreaseAllowance(IERC20 token, address spender, u
554         unchecked {
555             uint256 oldAllowance = token.allowance(address(this), s
556             require(oldAllowance >= value, "SafeERC20: decreased al
557             _callOptionalReturn(token, abi.encodeWithSelector(token
```

- Here smart contract is checking that msg.sender is zettahash_beneficiary address, currenttime is bigger than nextClaimTime, amount is smaller than or equal to maximum allowed tokens.



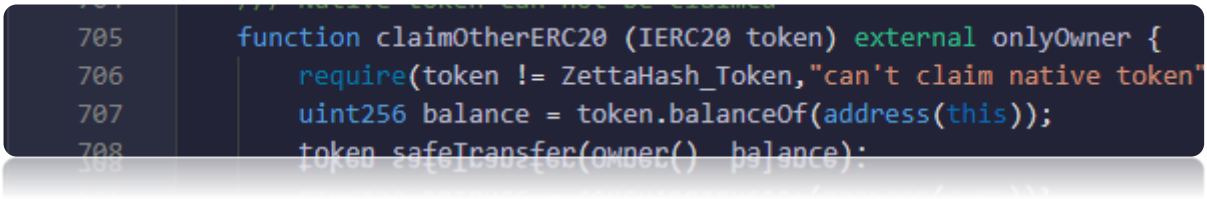
```
666     function claim (address to, uint256 amount) external {
667         require(msg.sender == Zettahash_Beneficiary, "Only zettahas
668         uint256 amountToSent = amount / 2;
```

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

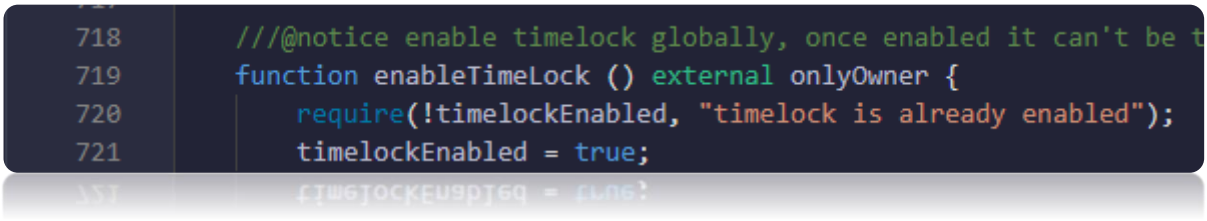


```
695     /// @param _newZetaBenificiary: new beneficiary to be assigned
696     function updateZettaHashHoldings (address _newZetaBenificiary)
697         require(_newZetaBenificiary != address(0),"error: zero addr
698         Zettahash_Beneficiary = _newZetaBenificiary;
699     }
```

- Here smart contract is checking that token address is not same as ZettaHash_token address.



- Here smart contract is checking that timelock is already enabled or not.



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.

Filename: ZH.sol

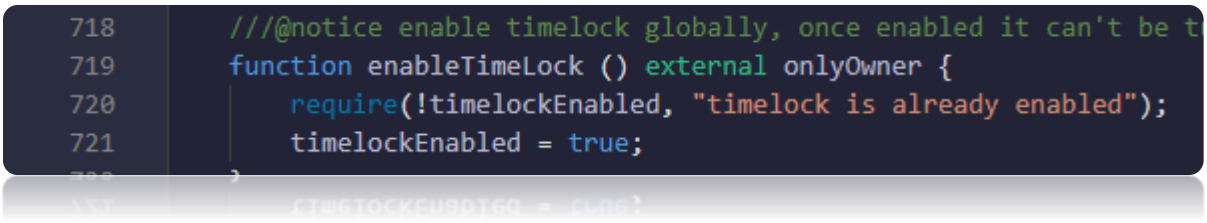
- **Function: - _approve**

```
518     function _approve(address owner, address spender, uint256 amount
519         require(owner != address(0), "ERC20: approve from the zero
520         require(spender != address(0), "ERC20: approve to the zero
521
522         _allowances[owner][spender] = amount;
523         emit Approval(owner, spender, amount);
524     }
```

- Here in _approve function you can check that owner has sufficient balance for giving allowance.

Filename: ZettahashHoldings.sol

- **Function: - enabledTimeLock**



```
718     ///@notice enable timelock globally, once enabled it can't be t
719     function enableTimeLock () external onlyOwner {
720         require(!timelockEnabled, "timelock is already enabled");
721         timelockEnabled = true;
```

- Here in enabledTimeLock function you are making timelockEnabled to true. But currently we don't have way to timelockEnabled false.

- I think it is good to have some functionality to make it false. Because zettahash_beneficiary address will not able to call claim method in emergency time.

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 excepted.

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