March 1 , 2022

Zcode NFT

SMART CONTRACT AUDIT

BY-SHEIKH AMAN (BLOCKCHAIN DEVELOPER AND AUDITOR)

Name

ZcodeNFT.sol

Time

1st March, 2022

For

Zcode Doge NFT

By

Content

Disclaimer

THIS REPORT DO NOT PROVIDE ANY WARRANTY OR GUARANTEE REGARDING THE ABSOLUTE BUG-FREE NATURE OF THE TECHNOLOGY ANALYZED. ANY INVESTMENT DECISION SHOULD NOT BE TAKEN BASED ON THIS REPORT AND IT DO NOT PROVIDE ANY INDICATION OF LEGAL COMPLIANCE.

About Me

Myself Sheikh Aman (SA audits), a freelancer. I have an experienced NodeJS backend development and Blockchain background. I prepare an Audit report for Smart Contract.

LinkedIn - https://www.linkedin.com/in/amsten/

About Zcode NFT contract

This is an NFT contract name "Zcode Doge NFT", symbol "ZCODE". It has a total supply of 1050 NFTs. It has 3 categories "GOLD, PLATINUM, LEGENDARY"

Checked Vulnerabilities

The code of this project has been taken from:

https://etherscan.io/address/0x82c0660d72ca6c381fecd19576c7bd6e4a38dbef#code

We have checked this smart contract for commonly known and more specific vulnerabilities. Following are those checked vulnerabilities.

- Solidity compiler version issue
- Variable shadowing
- Hardcoded addresses
- Reentrancy
- Business logics
- Gas limit and loops

Checked Vulnerabilities

- Malicious libraries
- Inappropriate use of access specifiers
- Unused codes or variables
- Contract size.
- Locked funds issue
- Naming conventions issues
- Boolean equality
- Unchecked math
- Missing error statements
- Changing state of variables after external calls
- tx.origin VS msg.sender
- mapping vs arrays
- Accessibility to restricted functions
- Security of on-chain data
- Uncheck sends
- Zero address checks

Severity Definitions

•High severity issues

High severity vulnerabilities are usually straightforward to exploit.

•Medium level severity issues

Medium level severity issues are important to fix but they are not straightforward to exploit

• Low-level severity issues

Low-level severity issues mostly contain unused codes, warnings etc they mostly don't affect the functionality of the smart contract but if fixed can optimize or reduce the size of the contract.

•Informational/Suggestion

These are basically related to coding practice, naming conventions etc if these are unfixed then also it will not affect smart contract actions.

Automated Testing

Slither

```
Low level call in Address.sendValue(address,uint256) (@openzeppelin\contracts\utils\Address.sol#55-60):
        - (success) = recipient.call{value: amount}() (@openzeppelin\contracts\utils\Address.sol#58)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin\contracts\utils\Address.sol#123-134):
        - (success,returndata) = target.call{value: value}(data) (@openzeppelin\contracts\utils\Address.sol#132)
Low level call in Address.functionStaticCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#152-161):
        - (success,returndata) = target.staticcall(data) (@openzeppelin\contracts\utils\Address.sol#159)
Low level call in Address.functionDelegateCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#179-188):
        - (success,returndata) = target.delegatecall(data) (@openzeppelin\contracts\utils\Address.sol#186)
Low level call in ZcodeNFT.withdraw() (CrashGame.sol#77-85):
        - (success1) = receiver.call{value: sh97}() (CrashGame.sol#81)
        - (success2) = dev.call{value: sh3}() (CrashGame.sol#83)
```

```
Low level call in Address.sendValue(address,uint256) (@openzeppelin\contracts\utils\Address.sol#55-60):
        - (success) = recipient.call{value: amount}() (@openzeppelin\contracts\utils\Address.sol#58)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin\contracts\utils\Address.sol#123-134):
        - (success,returndata) = target.call{value: value}(data) (@openzeppelin\contracts\utils\Address.sol#132)
Low level call in Address.functionStaticCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#152-161):
        - (success,returndata) = target.staticcall(data) (@openzeppelin\contracts\utils\Address.sol#159)
Low level call in Address.functionDelegateCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#179-188):
        - (success,returndata) = target.delegatecall(data) (@openzeppelin\contracts\utils\Address.sol#186)
Low level call in ZcodeNFT.withdraw() (CrashGame.sol#77-85):
        - (success1) = receiver.call{value: sh97}() (CrashGame.sol#81)
        - (success2) = dev.call{value: sh3}() (CrashGame.sol#83)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
ZcodeNFT.dev (CrashGame.sol#11) should be constant
ZcodeNFT.maxSupply (CrashGame.sol#29) should be constant
ZcodeNFT.receiver (CrashGame.sol#9-10) should be constant
ZcodeNFT.txLimit (CrashGame.sol#47) should be constant
```

Summary

We studied the given smart contracts performed some manual testing as well as automated checks using Slither and remix IDE. The purpose here was to find all the known coding bugs and then manually verify the issue reported by the automated tools. We further manually reviewed business logic and placed Defi-related aspects under scrutiny to find possible bugs.
So here is a list of all the issues found in the given ZcodeNFT.sol smart contract.

After checking and testing manually and deploying on test network, it has been observed that no serious bugs have been found. It doesn't contain any issue or error which effects business requirements.

Conclusion

The overall contract looks fine, as per us there is no bug in this, but as we mentioned at the beginning that auditing doesn't guarantee a totally bug-free system, so we recommend doing proper testing on testnet before going to mainnet.