BLOCKHAT
SECURITY

Wolf.Ai

Smart Contract Security Audit

Prepared by BlockHat

April 20th, 2023 - April 21st, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	Wolf Ecosystem
Version	0.1
Classification	Public

Scope

The Wolf.Ai Contract in the Wolf.Ai Repository

Link	Address
https://bscscan.com/address/0x925715C8d1E472d1B324fdAe6766Bb9c35381742#code	0x925715C8d1E472d1B324fdAe6766Bb9c35381742

Files	MD5 Hash
/Wolfcoin.sol	e03f554476c72db0a9655fd0b8d6a70c

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

Wolf.Ai engaged BlockHat to conduct a security assessment on the Wolf.Ai beginning on April 20th, 2023 and ending April 21st, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Wolf.Ai

Wolf.Ai is the leading artificial intelligence-based video generator that can be used for marketing, entertainment, and education and Business purposes. We pride ourselves on providing our customers with high-quality videos that are tailored to their exact needs. We use the latest technologies and algorithms to create videos that are both visually appealing and natural-looking. At Wolf.Ai , we strive to provide our customers with the best customer service experience possible. We understand that time is of the essence when it comes to video creation, and we work hard to provide our customers with the quickest turnaround times possible. We also offer customizable video packages to ensure that our customers get exactly what they need. We look forward to providing our customers with the best AI video generation services in the world.

Issuer	Wolf Ecosystem
Website	www.aiwolf.co
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

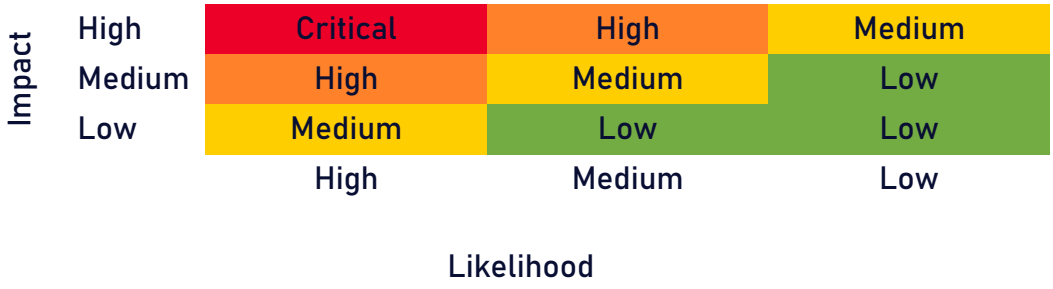
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Wolf.Ai implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include vulnerabilities.

Vulnerabilities	Severity	Status

3 Finding Details

A Wolfcoin.sol

No vulnerabilities found.

4 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
Wolfcoin._transfer(address,address,uint256) (Wolfcoin.sol#669-683)
   ↪ performs a multiplication on the result of a division:
       -burnfee = (amount.mul(2)).div(100) (Wolfcoin.sol#676)
       -super._transfer(from,to,amount - (2 * burnfee + wolffee)) (
          ↪ Wolfcoin.sol#678)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #divide-before-multiply

SafeMath.add(uint256,uint256) (Wolfcoin.sol#83-85) is never used and
   ↪ should be removed
SafeMath.div(uint256,uint256,string) (Wolfcoin.sol#162-167) is never
   ↪ used and should be removed
SafeMath.sub(uint256,uint256) (Wolfcoin.sol#97-99) is never used and
   ↪ should be removed
SafeMath.sub(uint256,uint256,string) (Wolfcoin.sol#143-148) is never
   ↪ used and should be removed
SafeMath.tryAdd(uint256,uint256) (Wolfcoin.sol#23-29) is never used and
   ↪ should be removed
SafeMath.tryDiv(uint256,uint256) (Wolfcoin.sol#65-70) is never used and
   ↪ should be removed
```

```
SafeMath.tryMul(uint256,uint256) (Wolfcoin.sol#48-58) is never used and
    ↪ should be removed
SafeMath.trySub(uint256,uint256) (Wolfcoin.sol#36-41) is never used and
    ↪ should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version0.8.17 (Wolfcoin.sol#8) necessitates a version too recent
    ↪ to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

Variable Wolfcoin.SHIB_BURN_WALLET (Wolfcoin.sol#649) is not in
    ↪ mixedCase
Variable Wolfcoin.Wolf_Treasury (Wolfcoin.sol#650) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions

Wolfcoin.SHIB_BURN_WALLET (Wolfcoin.sol#649) should be constant
Wolfcoin.Wolf_Treasury (Wolfcoin.sol#650) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-constant

transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Wolfcoin.sol#299-302)
name() should be declared external:
        - ERC20.name() (Wolfcoin.sol#340-342)
symbol() should be declared external:
        - ERC20.symbol() (Wolfcoin.sol#348-350)
totalSupply() should be declared external:
        - ERC20.totalSupply() (Wolfcoin.sol#372-374)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (Wolfcoin.sol#379-381)
```

```
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (Wolfcoin.sol#391-395)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Wolfcoin.sol#414-418)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (Wolfcoin.sol
          ↪ #436-441)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (Wolfcoin.sol#455-459)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (Wolfcoin.sol#475-484)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
Wolfcoin.sol analyzed (7 contracts with 78 detectors), 25 result(s)
    ↪ found
```

Conclusion:

Most of the vulnerabilities found by the analysis are false positive findings.

5 Conclusion

In this audit, we examined the design and implementation of Wolf.Ai contract and have not found any issue. The present code base is well-structured and ready for the mainnet.

BLOCKHAT
SECURITY

For a Contract Audit, contact us at contact@blockhat.io