CAT

Smart Contract Security Audit

Prepared by BlockHat

March 8th, 2023 - March 9th, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	CatArmy
Version	0.1
Classification	Public

Scope

The CAT Contract in the CAT Repository

Link	Address
https://bscscan.com/token/0x0173295183685f27c84db046b5f0bea3e683c24b	0x0173295183685F27C84db046B5F0bea3e683c24b

Files	MD5 Hash
StandardToken.sol	79da3a1f0831a9f453055f99b180d96c

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

CAT engaged BlockHat to conduct a security assessment on the CAT beginning on March 8th, 2023 and ending March 9th, 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 CAT

-

Issuer	CatArmy
Website	`www.catcattoken.com`
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.

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
			Likelihood	

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the CAT 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 , 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Avoid using .transfer() to transfer Ether	LOW	Not Fixed
Missing address verification	LOW	Not Fixed
Floating Pragma	LOW	Not Fixed

3 Finding Details

A StandardToken.sol

A.1 Avoid using .transfer() to transfer Ether [LOW]

Description:

Although transfer() and send() are recommended as a security best-practice to prevent reentrancy attacks because they only forward 2300 gas, the gas repricing of opcodes may break deployed contracts.

Code:

Listing 1: StandardToken.sol

```
constructor(
    string memory name_,
    string memory symbol_,
    uint8 decimals_,
    uint256 totalSupply_,
    address serviceFeeReceiver_,
    uint256 serviceFee_
) payable {
    _name = name_;
    _symbol = symbol_;
    _decimals = decimals_;
    _mint(owner(), totalSupply_);

    emit TokenCreated(owner(), address(this), TokenType.standard,
        ↪ VERSION);

    payable(serviceFeeReceiver_).transfer(serviceFee_);
}
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Consider using .call value: ... ("") instead, without hardcoded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

Status - Not Fixed

A.2 Missing address verification [LOW]

Description:

The address-type argument should serviceFeeReceiver_ include a zero-address test, otherwise, the contract owner will send ether to 0 address.

Code:

Listing 2: StandardToken.sol

```
    constructor(
        string memory name_,
        string memory symbol_,
        uint8 decimals_,
        uint256 totalSupply_,
        address serviceFeeReceiver_,
        uint256 serviceFee_
    ) payable {
        _name = name_;
        _symbol = symbol_;
        _decimals = decimals_;
        _mint(owner(), totalSupply_);
```

```
        emit TokenCreated(owner(), address(this), TokenType.standard,
           ↪ VERSION);

        payable(serviceFeeReceiver_).transfer(serviceFee_);
    }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

We recommend that you make sure the address provided in the argument are different from the address(0).

Status - Not Fixed

A.3 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.4. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensuring that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Code:

Listing 3: StandardToken.sol

```
pragma solidity ^0.8.4;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma not be used in production.Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status – Not Fixed

4 Best Practices

BP.1 Unused code

Description:

_msgData and _burn are not used in the smart contract, and make the code's review more difficult, we recommend to remove them.

Code:

Listing 4: StandardToken.sol

```
    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
```

Listing 5: StandardToken.sol

```
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address
            ↪ ");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(
            amount,
            "ERC20: burn amount exceeds balance"
        );
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
```

BP.2 Public functions can be external

Description:

Functions with a public scope that are not called inside the contract should be declared external to reduce the gas fees

Code:

Listing 6: StandardToken.sol

```
    function increaseAllowance(address spender, uint256 addedValue)
        public
        virtual
        returns (bool)
    {
        _approve(
            _msgSender(),
            spender,
            _allowances[_msgSender()][spender].add(addedValue)
        );
        return true;
    }
```

Listing 7: StandardToken.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue)
        public
        virtual
        returns (bool)
    {
        _approve(
            _msgSender(),
            spender,
            _allowances[_msgSender()][spender].sub(
                subtractedValue,
                "ERC20: decreased allowance below zero"
```

```
            )
        );
        return true;
    }
```

5 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:

```
StandardToken.allowance(address,address).owner (StandardToken.sol#259)
    ↪ shadows:
        - Ownable.owner() (StandardToken.sol#167-169) (function)
StandardToken._approve(address,address,uint256).owner (StandardToken.sol
    ↪ #367) shadows:
        - Ownable.owner() (StandardToken.sol#167-169) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #local-variable-shadowing

StandardToken.constructor(string,string,uint8,uint256,address,uint256).
    ↪ serviceFeeReceiver_ (StandardToken.sol#210) lacks a zero-check on
    ↪  :
                - address(serviceFeeReceiver_).transfer(serviceFee_) (
                    ↪ StandardToken.sol#220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation

Context._msgData() (StandardToken.sol#111-113) is never used and should
    ↪ be removed
```

```
SafeMath.div(uint256,uint256) (StandardToken.sol#64-66) is never used
    ↪ and should be removed
SafeMath.div(uint256,uint256,string) (StandardToken.sol#83-92) is never
    ↪ used and should be removed
SafeMath.mod(uint256,uint256) (StandardToken.sol#68-70) is never used
    ↪ and should be removed
SafeMath.mod(uint256,uint256,string) (StandardToken.sol#94-103) is never
    ↪  used and should be removed
SafeMath.mul(uint256,uint256) (StandardToken.sol#60-62) is never used
    ↪ and should be removed
SafeMath.sub(uint256,uint256) (StandardToken.sol#56-58) is never used
    ↪ and should be removed
SafeMath.tryAdd(uint256,uint256) (StandardToken.sol#14-20) is never used
    ↪  and should be removed
SafeMath.tryDiv(uint256,uint256) (StandardToken.sol#38-43) is never used
    ↪  and should be removed
SafeMath.tryMod(uint256,uint256) (StandardToken.sol#45-50) is never used
    ↪  and should be removed
SafeMath.tryMul(uint256,uint256) (StandardToken.sol#29-36) is never used
    ↪  and should be removed
SafeMath.trySub(uint256,uint256) (StandardToken.sol#22-27) is never used
    ↪  and should be removed
StandardToken._burn(address,uint256) (StandardToken.sol#353-364) is
    ↪ never used and should be removed
StandardToken._setupDecimals(uint8) (StandardToken.sol#378-380) is never
    ↪  used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Variable StandardToken._totalSupply (StandardToken.sol#203) is too
    ↪ similar to StandardToken.constructor(string,string,uint8,uint256,
    ↪ address,uint256).totalSupply_ (StandardToken.sol#209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-are-too-similar
```

```
renounceOwnership() should be declared external:
	- Ownable.renounceOwnership() (StandardToken.sol#176-178)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (StandardToken.sol#180-183)
name() should be declared external:
	- StandardToken.name() (StandardToken.sol#223-225)
symbol() should be declared external:
	- StandardToken.symbol() (StandardToken.sol#227-229)
decimals() should be declared external:
	- StandardToken.decimals() (StandardToken.sol#231-233)
totalSupply() should be declared external:
	- StandardToken.totalSupply() (StandardToken.sol#235-237)
balanceOf(address) should be declared external:
	- StandardToken.balanceOf(address) (StandardToken.sol#239-247)
transfer(address,uint256) should be declared external:
	- StandardToken.transfer(address,uint256) (StandardToken.sol
	  ↪ #249-257)
allowance(address,address) should be declared external:
	- StandardToken.allowance(address,address) (StandardToken.sol
	  ↪ #259-267)
approve(address,uint256) should be declared external:
	- StandardToken.approve(address,uint256) (StandardToken.sol
	  ↪ #269-277)
transferFrom(address,address,uint256) should be declared external:
	- StandardToken.transferFrom(address,address,uint256) (
	  ↪ StandardToken.sol#279-294)
increaseAllowance(address,uint256) should be declared external:
	- StandardToken.increaseAllowance(address,uint256) (StandardToken
	  ↪ .sol#296-307)
decreaseAllowance(address,uint256) should be declared external:
	- StandardToken.decreaseAllowance(address,uint256) (StandardToken
	  ↪ .sol#309-323)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
StandardToken.sol analyzed (6 contracts with 78 detectors), 31 result(s)
    ↪  found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

We examined the design and implementation of CAT in this audit and found several issues of various severities. We advise CatArmy team to implement the recommendations contained in all 3 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

BLOCKHAT SECURITY

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