



BLOCKHAT
SECURITY

MTHB

Smart Contract Security Audit

Prepared by BlockHat

March 19th, 2025 – March 21st, 2025

BlockHat.io

contact@blockhat.io

Document Properties

Client	MTHB
Version	0.1
Classification	Public

Scope

The MTHB smart contracts (Audit and Re-audit smart contracts)

Link	Address
https://bscscan.com/token/0xbb1F4773058357C34D2EF07759f17fc55635C2Ad	0xbb1F4773058357C34D2EF07759f17fc55635C2Ad

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1	Introduction	4
1.1	About MTHB	4
1.2	Approach & Methodology	4
1.2.1	Risk Methodology	5
2	Findings Overview	6
2.1	Summary	6
2.2	Key Findings	6
3	Finding Details	7
A	MTHB.sol	7
A.1	Unchecked Transfer of Tokens in <code>rescueToken</code> [MEDIUM]	7
A.2	ERC-20 Approval Race Condition Vulnerability [LOW]	8
A.3	Floating Pragma Version [LOW]	10
A.4	Use of Outdated ERC20 Implementation [INFORMATIONAL]	11
4	Static Analysis (Slither)	12
5	Conclusion	15

1 Introduction

MTHB engaged BlockHat to conduct a security assessment on the MTHB beginning on March 19th, 2025 and ending March 21st, 2025. 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 MTHB

Issuer	MTHB
Website	https://mthaibaht.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		Likelihood		
		High	Medium	Low
	High	Critical	High	Medium
	Medium	High	Medium	Low
Low	Low	Medium	Low	Low
		High	Medium	Low

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the MTHB 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 , **1** medium-severity, **2** low-severity, **1** informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Unchecked Transfer of Tokens in <code>rescueToken</code>	MEDIUM	Not fixed
ERC-20 Approval Race Condition Vulnerability	LOW	Not fixed
Floating Pragma Version	LOW	Not fixed
Use of Outdated ERC20 Implementation	INFORMATIONAL	Not fixed

3 Finding Details

A MTHB.sol

A.1 Unchecked Transfer of Tokens in `rescueToken` [MEDIUM]

Description:

The `rescueToken` function uses the `transfer` function to send tokens without verifying the return value. Some ERC-20 tokens (e.g., USDT) do not adhere to the ERC-20 standard strictly and may not revert on failure, leading to potential loss of funds if the transfer fails silently.

Listing 1: MAKECOINGENRATOR.sol

```
592 function rescueToken(address tokenAddress , address receiver) public  
    ↪ onlyOwner {  
593     uint balance = IERC20(tokenAddress).balanceOf(address(this));  
594     IERC20(tokenAddress).transfer(receiver, balance);  
595 }
```

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

Use OpenZeppelin's `SafeERC20.safeTransfer` or manually check the return value of `transfer`. This ensures that the transfer succeeds and mitigates the risk of funds being lost due to silent failures.

Status - Not fixed

A.2 ERC-20 Approval Race Condition Vulnerability [LOW]

Description:

The `approve` function in the provided ERC-20 token contract allows the owner to set an allowance for a spender without enforcing a safe workflow to prevent a race condition. If an owner increases an existing allowance (e.g., from 100 to 200 tokens), a malicious spender could execute `transferFrom` with the old allowance (100 tokens) before the new approval is processed and then use the new allowance (200 tokens) afterward, effectively spending more tokens than intended (up to 300 tokens in this example). This issue is acknowledged in the IERC20 interface comments, which recommend reducing the allowance to 0 before setting a new value, but the contract does not enforce this mitigation.

Listing 2: MAKECOINGENRATOR.sol

```
242     function approve(address spender, uint256 amount)
243         public
244         virtual
245         override
246         returns (bool)
247     {
248         _approve(_msgSender(), spender, amount);
249         return true;
250     }
```

Listing 3: MAKECOINGENRATOR.sol

```
408     function _approve(
409         address owner,
410         address spender,
411         uint256 amount
412     ) internal virtual {
413         require(owner != address(0), "ERC20: approve from the zero
            ↳ address");
414         require(spender != address(0), "ERC20: approve to the zero
            ↳ address");
```



```

416         _allowances[owner][spender] = amount;
417         emit Approval(owner, spender, amount);
418     }

```

Risk Level:

Likelihood – 2

Impact – 2

Recommendation:

1. Enforce a “zero-first” approval workflow by requiring the allowance to be set to 0 before a new non-zero value is approved (e.g., using a custom [safeApprove](#) function). 2. Use the [increaseAllowance](#) and [decreaseAllowance](#) functions exclusively for adjusting allowances, as these functions are already implemented and reduce the risk by performing atomic updates. For example, add a check in [approve](#) to revert if the current allowance is non-zero and the new amount is non-zero, or document that users should rely on [increaseAllowance/decreaseAllowance](#) instead of [approve](#).

Listing 4: Proposed Safe Approve Function

```

1     function approve(address spender, uint256 amount) public virtual
      ↳ override returns (bool) {
2         require(
3             _allowances[_msgSender()][spender] == 0 & amount == 0,
4             "ERC20: approve race condition risk - use increaseAllowance/
      ↳ decreaseAllowance or reset to 0 first"
5         );
6         _approve(_msgSender(), spender, amount);
7         return true;
8     }

```

Alternatively, promote the use of:

Listing 5: MAKECOINGENRATOR.sol

```
279     function increaseAllowance(address spender, uint256 addedValue)
        ↳ public virtual returns (bool)
```

Listing 6: MAKECOINGENRATOR.sol

```
298     function decreaseAllowance(address spender, uint256 subtractedValue)
        ↳ public virtual returns (bool)
```

Status - Not fixed

A.3 Floating Pragma Version [LOW]

Description:

The smart contract uses a floating Solidity pragma (0.8.19). This means the compiler may use any minor version within the 0.8.x range, potentially leading to unexpected behavior if a newer minor version introduces changes or deprecations. Using a fixed Solidity version enhances security by ensuring consistency in compilation.

Listing 7: MAKECOINGENRATOR.sol

```
11 pragma solidity ^0.8.19;
```

Risk Level:

Likelihood - 1

Impact - 2

Recommendation:

Specify a fixed Solidity version (e.g., `pragma solidity 0.8.19;`) to ensure consistent compilation and avoid potential issues from unintended compiler updates. Always verify that the chosen version is the most suitable for security and functionality.

Status - Not fixed

A.4 Use of Outdated ERC20 Implementation [INFORMATIONAL]

Description:

The contract implements its own version of the ERC20 standard, which may not include the latest security practices, optimizations, and features found in widely-used libraries such as OpenZeppelin's ERC20 implementation. Using an outdated or custom implementation can introduce risks and compatibility issues with other contracts and decentralized applications (dApps).

Recommendation:

Replace the custom ERC20 implementation with the latest version of the ERC20 token standard from a reputable library such as OpenZeppelin. This not only ensures compliance with the latest security practices but also benefits from the community's scrutiny, ongoing maintenance, and updates.

Status - Not fixed

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:

```
MAKECOINGENRATOR.rescueToken(address,address) (token.sol#596-599)
  ↳ ignores return value by IERC20(tokenAddress).transfer(receiver,
  ↳ balance) (token.sol#598)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
  ↳ #unchecked-transfer

INFO:Detectors:
ServicePayer.constructor(address,string,address) (token.sol#562-564)
  ↳ ignores return value by IPayable(receiver).payServicesFees{value:
  ↳ msg.value}(serviceName,_refaddress) (token.sol#563)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
  ↳ #unused-return

INFO:Detectors:
MAKECOINGENRATOR.constructor(string,string,uint8,uint256,address,string,
  ↳ address)._name (token.sol#569) shadows:
- ERC20._name (token.sol#131) (state variable)
MAKECOINGENRATOR.constructor(string,string,uint8,uint256,address,string,
  ↳ address)._symbol (token.sol#570) shadows:
- ERC20._symbol (token.sol#132) (state variable)
MAKECOINGENRATOR.constructor(string,string,uint8,uint256,address,string,
  ↳ address)._decimals (token.sol#571) shadows:
- ERC20._decimals (token.sol#133) (state variable)
```

MAKECOINGENRATOR.constructor(string,string,uint8,uint256,address,string,
 ↳ address)._totalSupply (token.sol#572) shadows:

- ERC20._totalSupply (token.sol#129) (state variable)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation>
 ↳ #local-variable-shadowing

INFO:Detectors:

MAKECOINGENRATOR.rescueBalance(address).toAddr (token.sol#591) lacks a
 ↳ zero-check on :

- (success,None) = toAddr.call{value: address(this).balance}() (token.
 ↳ sol#592)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation>
 ↳ #missing-zero-address-validation

INFO:Detectors:

Context._msgData() (token.sol#18-20) is never used and should be removed

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation>
 ↳ #dead-code

INFO:Detectors:

Version constraint ^0.8.19 contains known severe issues (<https://solidity.readthedocs.io/en/latest/bugs.html>)
 ↳ solidity.readthedocs.io/en/latest/bugs.html

- VerbatimInvalidDeduplication
- FullInlinerNonExpressionSplitArgumentEvaluationOrder
- MissingSideEffectsOnSelectorAccess.

It is used by:

- ^0.8.19 (token.sol#11)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation>
 ↳ #incorrect-versions-of-solidity

INFO:Detectors:

Low level call in MAKECOINGENRATOR.rescueBalance(address) (token.sol
 ↳ #591-594):

- (success,None) = toAddr.call{value: address(this).balance}() (token.
 ↳ sol#592)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation>
 ↳ #low-level-calls

INFO:Detectors:

```
ERC20._decimals (token.sol#133) should be immutable
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
```

```
↪ #state-variables-that-could-be-declared-immutable
```

```
INFO:Slither:token.sol analyzed (8 contracts with 100 detectors), 11
```

```
↪ result(s) found
```

Conclusion:

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

5 Conclusion

We examined the design and implementation of MTHB in this audit and found several issues of various severities. We advise MTHB team to implement the recommendations contained in all 4 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 Smart Contract Audit, contact us at contact@blockhat.io