BLOCKSEC

Security Audit

Report for ERC20TokenWrapped Contract

Date: April 4th, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About Target Contracts 1
1.2 Disclaimer 1
1.3 Procedure of Auditing 2
1.3.1 Software Security 2
1.3.2 DeFi Security 2
1.3.3 NFT Security 2
1.3.4 Additional Recommendation 3
1.4 Security Model 3

Chapter 2 Findings **4**
2.1 Note 4
2.1.1 Potential single point of failures 4
2.1.2 Potential centralization risks 4

Report Manifest

Item	Description
Client	Merlin Chain
Target	ERC20TokenWrapped Contract

Version History

Version	Date	Description
1.0	April 4th, 2024	First Release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

Email: audit@blocksec.com

Date: 2024-04-06 15:51:55+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The focus of this audit is on the ERC20TokenWrapped Contract. The contract was deployed and verified at the address 0x5c46bFF4B38dc1EAE09C5BAc65872a1D8bc87378 in Merlin chain [1].

This contract implements a ERC20 token with permit and capacity. It uses the OZ `ERC20Permit` and `ERC20Capped` library. However, the functions including `mint` and `burn` are privileged, which can only be invoked through a specified bridge address. This address is 0xa212d68499947960 cd3A24861E788E7C38c0fb9D when preparing this report.

The auditing process is iterative. Specifically, we would audit the commits (or deployed addresses) that fix the discovered issues. If there are new issues, we will continue this process. The deployed addresses during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Deployed Address
ERC20TokenWrapped Contract	`Version 1`	`0x5c46bFF4B38dc1EAE09C5BAc65872a1D8bc87378`
	`Version 2`	-

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly

[1] https://scan.merlinchain.io/address/0x5c46bFF4B38dc1EAE09C5BAc65872a1D8bc87378

specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item

* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we do not find potential security issues. Besides, we also have **two** notes.

- Note: 2

ID	Severity	Description	Category	Status
1	-	Potential single point of failures	Note	-
2	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Note

2.1.1 Potential single point of failures

Description In the ERC20TokenWrapped Contract, the bridge has privileged capabilities to mint and burn the tokens. The bridge address is 0xa212d68499947960cd3a24861e788e7c38c0fb9d, a bridge contract in the Merlin chain.

This may introduce a single point of failure. For instance, if the accounts that can invoke functions in the bridge are compromised, then the tokens can be minted or burned. The project should take this into consideration into daily operation, and deploy a monitoring system to actively monitor privileged behaviors.

Feedback from the Project The bridge has a whitelist to control the addresses that can invoke the mint function. We use the cobo MPC to secure the private keys to invoke this function. For the burn function, the bridge ensures that one can only burn his own tokens.

2.1.2 Potential centralization risks

Description In the ERC20TokenWrapped Contract, there is blacklist that prevents the addresses in this list to send tokens. This may cause the centralization concern. The project should carefully maintain this blacklist and only add necessary ones for compliance purpose.

Feedback from the Project The backlist can only be added by a multisig address. We will carefully add the addresses to blacklist.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS