BLOCKSEC

Security Audit Report for Metapool-Shortcut

Date: May 27, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Metapool
Target	Metapool-Shortcut

Version History

Version	Date	Description
1.0	May 27, 2024	First release

Signature

Chapter 1 Introduction

1.1 About Target Contracts

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

The target of this audit is the code repository of Metapool-Shortcut[1] of Metapool. Note that the metapool-shortcut protocol relies on the external contracts Bridge and mpETH. The security issues of these contracts are beyond the scope of the audit.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values 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	Commit Hash
Metapool-Shortcut	`Version 1`	`3cccd7249461f6321e7715e613d3651fd1a97c04`
Metapool-Shortcut	`Version 2`	`65ec2d7d0e2031640014139dbd6238ab306c7cb2`

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 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] https://github.com/Meta-Pool/metapool-shortcut

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 find **three** recommendations. Besides, we also have **one** notes.

- Recommendation: 3
- Note: 1

ID	Severity	Description	Category	Status
1	-	Redundant code	Recommendation	Fixed
2	-	Lack of check on the address variables	Recommendation	Fixed
3	-	Lack of invocation of disableInitializers() in the implementation contract	Recommendation	Fixed
4	-	Pontential centralization risk	Note	-

The details are provided in the following sections.

2.1 Additional Recommendation

2.1.1 Redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `SwapToMpEthOnLineaV1`, the `error UnsuccessfulApproval()` and the variable `chainId` were not used.

```
12 contract SwapToMpEthOnLineaV1 is Initializable, OwnableUpgradeable {
13   using SafeERC20 for IERC20;
14
15   uint256 public chainId;
16   address public bridge;
17   address public mpeth;
18
19   uint256 public constant COMPLEXITY = 2;
20   uint256 public constant BASE_FEE = 1000;
21
22   error LessThanMinValue();
23   error UnsuccessfulApproval();
```

Listing 2.1: SwapToMpEthOnLineaV1.sol

Suggestion Remove the redundant code.

2.1.2 Lack of check on the address variables

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `initialize()`, there is no check to ensure that the `_bridge, _mpeth` and `_owner` addresses are not `address(0)`. Additionally, the function `updateBridgeAddress()`

lacks a check to ensure that the `_bridge` address is not `address(0)` and the new `_bridge` address is different from the previous one.

```
28 function initialize(
29     uint256 _chainId,
30     address _bridge,
31     address _mpeth,
32     address _owner
33 ) public initializer {
34     __Ownable_init(_owner);
35     chainId = _chainId;
36     bridge = _bridge;
37     mpeth = _mpeth;
38 }
```

Listing 2.2: SwapToMpEthOnLineaV1.sol

```
40 function updateBridgeAddress(address _bridge) external onlyOwner returns (bool) {
41     bridge = _bridge;
42     return true;
43 }
```

Listing 2.3: SwapToMpEthOnLineaV1.sol

Suggestion Add the check accordingly.

2.1.3 Lack of invocation of disableInitializers() in the implementation contract

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `SwapToMpEthOnLineaV1` is designed as an implementation contract. However, it lacks an invocation of `disableInitializers()` to prevent the initialization of the implementation contract.

Suggestion Add the invocation of the function `disableInitializers()`.

2.2 Note

2.2.1 Pontential centralization risk

Description There are some potential centralization risks in the protocol. Specifically, the privileged admin has the ability to update a few system variables (e.g., `bridge`,...) . If the private key is leaked, the users' assets can be lost.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS