BLOCKSEC

Security Audit
Report for mantle-v2

Date: October 26, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About the Audit Target 1
1.2 Disclaimer 1
1.3 Procedure of Auditing 2
1.3.1 Security Issues 2
1.3.2 Additional Recommendation 3
1.4 Security Model 3

Chapter 2 Findings **4**
2.1 Security Issue 4
2.1.1 Lack of upper bound on the withdrawal gas limit 4
2.2 Note 7
2.2.1 Excess gas limit when the upgrade is deployed before the Glamsterdam migration 7
2.2.2 Assumptions of gas budgets 7

Report Manifest

Item	Description
Client	Mantle
Target	mantle-v2

Version History

Version	Date	Description
1.0	October 26, 2026	First release

Signature



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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-10-29 04:41:03+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 the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of mantle-v2 of Mantle.

This upgrade aligns the `CrossDomainMessenger` gas budgets with the gas costs introduced in the Glamsterdam hard fork. The finalization reserve and the pre-call check buffer each rise to 150,000, and a new overhead of 185,000 gas is introduced to cover cold calls that create the recipient account and is folded into both `baseGas` and the `hasMinGas` reservation. All three `relayMessage()` implementations copy `_message` to memory at function entry rather than at the call site, so every budget is measured after the copy while the ABI, message hashes, and storage layout remain unchanged.

Note this audit only focuses on the codes located in the following directories/files.

- packages/contracts-bedrock/src/universal/CrossDomainMessenger.sol
- packages/contracts-bedrock/src/L1/L1CrossDomainMessenger.sol
- packages/contracts-bedrock/src/L2/L2CrossDomainMessenger.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

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. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
mantle-v2	`Version 0`	`2a3541c9ef0a641bb853e561e15135afb38aa2e3`
	`Version 1`	`8df5df7b871e2a3b8f297221cb860852f343893e`
	`Version 2`	`d34de734101af0b4252ddb77987267586c7f8352`

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,

[1] https://github.com/mantle-xyz/mantle-v2

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 Target, 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 the Target.

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.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target 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 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency

- Emergency mechanism
- Economic and incentive impact

1.3.2 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 five 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.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **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 found **one** potential security issue. Besides, we have **two** notes.

- Low Risk: 1
- Note: 2

ID	Severity	Description	Category	Status
1	Low	Lack of upper bound on the withdrawal gas limit	Security Issue	Fixed
2	-	Excess gas limit when the upgrade is deployed before the Glamsterdam migration	Note	-
3	-	Assumptions of gas budgets	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of upper bound on the withdrawal gas limit

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 0`

Description In the contract `L2CrossDomainMessenger`, the function `sendMessage()` passes a user-controlled `_minGasLimit` to function `baseGas()` to calculate the gas limit of the L2-to-L1 withdrawal. However, it does not enforce an upper bound to ensure this gas limit can be satisfied within the execution gas available during L1 finalization.

Glamsterdam caps execution gas at 2^{24} (16,777,216) gas. Because the function `OptimismPortal.finalizeWithdrawalTransaction()` is executed through the proxy's `DELEGATECALL`, EIP-150 withholds one sixty-fourth of the available gas. The function can therefore receive at most (16,777,216 * 63 / 64 = 16,515,072) gas, before counting intrinsic gas and all proxy execution costs.

For an empty payload and `_minGasLimit = 15,300,000`, function `baseGas()` produces a gas limit of 16,268,657. Accounting for EIP-150's forwarding rule and the additional 40,000 gas reserved by function `SafeCall.callWithMinGas()`, the required gas is $\lceil 16,268,657 * 64/63 \rceil$ + 40,000 = 16,566,890. This exceeds the maximum gas available to the function `finalizeWithdrawalTransaction()`, resulting in a revert on every finalization attempt.

```
109  function sendMessage(address _target, bytes calldata _message, uint32 _minGasLimit) external
         payable override {
110      require(_target != tx.origin || msg.value == 0, "once target is an EOA, msg.value must be
             zero");
111      require(_target != L1_MNT_ADDRESS, "target must not be MNT address on L1");
112
113      // Triggers a message to the other messenger. Note that the amount of gas provided to the
114      // message is the amount of gas requested by the user PLUS the base gas value. We want to
115      // guarantee the property that the call to the target contract will always have at least
```

```
116     // the minimum gas limit specified by the user.
117     _sendMessage(
118         0,
119         OTHER_MESSENGER,
120         baseGas(_message, _minGasLimit),
121         abi.encodeWithSelector(
122             L1CrossDomainMessenger.relayMessage.selector,
123             messageNonce(),
124             msg.sender,
125             _target,
126             msg.value,
127             0,
128             _minGasLimit,
129             _message
130         )
131     );
132
133     emit SentMessage(_target, msg.sender, _message, messageNonce(), _minGasLimit);
134     emit SentMessageExtension1(msg.sender, msg.value, 0);
135
136     unchecked {
137         ++msgNonce;
138     }
139 }
```

Listing 2.1: packages/contracts-bedrock/src/L2/L2CrossDomainMessenger.sol

```
116 function callWithMinGas(
117     address _target,
118     uint256 _minGas,
119     uint256 _value,
120     bytes memory _calldata
121 )
122     internal
123     returns (bool)
124 {
125     bool _success;
126     bool _hasMinGas = hasMinGas(_minGas, 0);
127     assembly {
128         // Assertion: gasleft() >= (_minGas * 64) / 63 + 40_000
129         if iszero(_hasMinGas) {
130             // Store the "Error(string)" selector in scratch space.
131             mstore(0, 0x08c379a0)
132             // Store the pointer to the string length in scratch space.
133             mstore(32, 32)
134             // Store the string.
135             //
136             // SAFETY:
137             // - We pad the beginning of the string with two zero bytes as well as the
138             // length (24) to ensure that we override the free memory pointer at offset
139             // 0x40. This is necessary because the free memory pointer is likely to
140             // be greater than 1 byte when this function is called, but it is incredibly
141             // unlikely that it will be greater than 3 bytes. As for the data within
```

```
142             // 0x60, it is ensured that it is 0 due to 0x60 being the zero offset.
143             // - It's fine to clobber the free memory pointer, we're reverting.
144             mstore(88, 0x08c379a0000000000000000000000000000000000000000000000000000000)
145
146             // Revert with 'Error("SafeCall: Not enough gas")'
147             revert(28, 100)
148         }
149
150         // The call will be supplied at least ((_minGas * 64) / 63) gas due to the
151         // above assertion. This ensures that, in all circumstances (except for when the
152         // `_minGas` does not account for the `memory_expansion_cost` and `code_execution_cost`
153         // factors of the dynamic cost of the `CALL` opcode), the call will receive at least
154         // the minimum amount of gas specified.
155         _success :=
156             call(
157                 gas(), // gas
158                 _target, // recipient
159                 _value, // ether value
160                 add(_calldata, 32), // inloc
161                 mload(_calldata), // inlen
162                 0x00, // outloc
163                 0x00 // outlen
164             )
165     }
166     return _success;
167 }
```

Listing 2.2: packages/contracts-bedrock/src/libraries/SafeCall.sol

```
100 function hasMinGas(uint256 _minGas, uint256 _reservedGas) internal view returns (bool) {
101     bool _hasMinGas;
102     assembly {
103         // Equation: gas × 63 ≥ minGas × 64 + 63(40_000 + reservedGas)
104         _hasMinGas := iszero(lt(mul(gas(), 63), add(mul(_minGas, 64), mul(add(40000, _reservedGas)
                , 63))))
105     }
106     return _hasMinGas;
107 }
```

Listing 2.3: packages/contracts-bedrock/src/libraries/SafeCall.sol

Impact An affected L2-to-L1 withdrawal cannot be finalized, leaving its associated assets stranded.

Suggestion Reject withdrawals in function `sendMessage()` when the complete destination-side gas requirement exceeds a bound.

Feedback from the project The project resolved this issue in PR #408 [1]. The fix adds checks to the function `L2CrossDomainMessenger._sendMessage()`, limiting the withdrawal gas limit and estimated L1 calldata gas cost to 15,000,000 gas, and the encoded relay data to 90,000 bytes. These limits accommodate L1 finalization costs and transaction-size constraints, assuming

[1] https://github.com/mantle-xyz/mantle-v2/pull/408

direct EOA type-2 transactions with empty access lists. The checks cover both `sendMessage()` overloads and bridge calls, and direct calls to contract `L2ToL1MessagePasser` remain outside this change.

2.2 Note

2.2.1 Excess gas limit when the upgrade is deployed before the Glamsterdam migration

Introduced by `Version 1`

Description The upgrade raises both `baseGas` and the gas reservation passed to the function `hasMinGas()`. These budgets are calibrated to the Glamsterdam gas schedule, which assigns higher costs to account creation and state growth than the schedule in effect before the migration.

If the upgrade is deployed before Glamsterdam activates, every relay executes against the older schedule while carrying oversized budgets. As a result, the `baseGas` quoted for each new cross-domain message is higher than the pre-upgrade value, and the function `hasMinGas()` demands the same additional amount. The condition is temporary and carries no security impact. However, users or relayers may need to provide a higher gas limit for finalization transactions. The project should therefore treat the upgrade and the Glamsterdam migration as a paired rollout.

Feedback from the project The project plans to complete the contract upgrade shortly before the L1 upgrade is activated, and will also inform users through its official documentation that the upgrade may increase transaction costs.

2.2.2 Assumptions of gas budgets

Introduced by `Version 1`

Description In contract `CrossDomainMessenger`, constants `RELAY_RESERVED_GAS`, `RELAY_GAS_CHECK_BUFFER`, and `RELAY_NEW_ACCOUNT_OVERHEAD` define gas budgets for different execution stages of function `relayMessage()`. Each budget should be estimated independently based on the applicable gas schedule and the execution costs of its corresponding stage. The project should validate each budget against actual gas consumption on both L1 and L2, accounting for their respective gas schedules and execution paths, and maintain sufficient margins as either layer's gas costs change.

Feedback from the project The project confirmed that it assessed each budget separately under the updated gas schedule. `RELAY_RESERVED_GAS` is set to 150,000 gas for post-call finalization, leaving a margin of approximately 36,000 gas in the measured proxy-token scenario when state-gas returns are excluded. `RELAY_GAS_CHECK_BUFFER` is set to 150,000 gas for the operations between the gas check and the target call, leaving a margin of approximately 14,000 gas in the measured cold-access scenario. `RELAY_NEW_ACCOUNT_OVERHEAD` adds 185,000 gas to the existing 40,000 gas call budget, providing 225,000 gas against an estimated maximum dynamic call cost of 197,900 gas.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS