MetisForge Security Review

Duration: January 29, 2025 - February 01, 2025

Date: February 7, 2025

Conducted by: **KeySecurity**
gkrastenov, Lead Security Researcher

Table of Contents

1 About KeySecurity

KeySecurity is a new, innovative Web3 security company that hires top-talented security researchers for your project. We have conducted over 30 security reviews for various projects, collectively holding over $300,000,000 in TVL. For security audit inquiries, you can reach out to us on Twitter/X or Telegram `@gkrastenov` or check our previous work `here`.

2 About MetisForge

`Metis Forge` is the leading platform for meme tokens on Metis. Experience instant, AI-assisted token launches on MetisL2.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

4 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	MetisForge
Contract Address	Metis Network: 0x3DFFcfc36B49687540E120c43d7B4Be4e6758531
Documentation	N/A
Methods	Manual review

Scope

MemeTokenFactory.sol
MemeToken.sol
MemeTokenRaise.sol

Timeline

January 29, 2025	Audit kick-off
February 01, 2025	Preliminary report
February 06, 2025	Mitigation review

Issues Found

Severity	**Count**
High	0
Medium	2
Low	3
Information	0
Total	**5**

6 Findings

6.1 Medium

6.1.1 Use .call() instead of .transfer() to transfer native tokens

Severity: *Medium*

Context: Global

Description: The protocol uses Solidity's `transfer()` when transferring Matis to the recipients. This has some notable shortcomings when the recipient is a smart contract, which can render Matis impossible to transfer. Specifically, the transfer will inevitably fail when the smart contract:

- does not implement a payable fallback function
- implements a payable fallback function which would incur more than 2300 gas units
- implements a payable fallback function incurring less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300

Recommendation: Use `.call()` to transfer native tokens instead.

Resolution and Client comment: Resolved.

6.1.2 If adding liquidity fails, the user can not refund their funds

Severity: *Medium*

Context: MemeTokenRaise.sol#365

Description: When the `addLiquidityAndLock` function is called by the owner to add liquidity, a try-catch mechanism is used. If everything works correctly, the logic continues in the `try` block, where the owner receives their token amount, and `liquidityAdded` is set to `true`. However, if `positionHelper.addLiquidityAndCreatePosition` fails, execution moves to the `catch` block, where the event `Log(reason)` is emitted.

This can be problematic for users if the issue preventing liquidity addition is not resolved. The owner must call the `enableRefunds` function to allow users to refund their funds. Unfortunately, in this case, users will not be able to refund their funds because all funds in the `MemeTokenRaise` contract are deposited in wrapped Metis, and the contract will not be able to transfer their funds.

```
    function refund() external nonReentrant {
        require(canRefund, "Refunds are not allowed");
        require(!liquidityAdded, "Liquidity already added");
        require(userDeposits[msg.sender] > 0, "No deposit to refund");

        uint256 depositAmount = userDeposits[msg.sender];
        userDeposits[msg.sender] = 0;
        totalDeposits -= depositAmount;

        payable(msg.sender).transfer(depositAmount);
        emit Refunded(msg.sender, depositAmount);
    }
```

Recommendation: Revert the transaction if `addLiquidityAndCreatePosition` fails.

Resolution and Client comment: Resolved.

6.2 Low

6.2.1 The token creator can block the adding of liquidity

Severity: *Low*

Context: MemeTokenRaise.sol#342

Description: When `addLiquidityAndLock` function is called, a reward is transferred to the token creator. As a reward for creating the meme token, every token creator receives a small amount. However, if the token creator is a malicious user or a smart wallet is used, the transfer of Metis to the token creator may fail, which could block the liquidity addition.

```
// Transfer creator reward in Metis
        payable(tokenCreator).transfer(creatorRewardMetis);
        emit CreatorRewarded(tokenCreator, creatorRewardMetis);
```

Recommendation: Use `.call`, and if the transfer fails, continue executing the logic.

Resolution and Client comment: Acknowledged.

6.2.2 catch Error(string memory reason) does not handle all possible errors

Severity: *Low*

Context: MemeTokenRaise.sol#392

Description: The `catch Error(string memory reason)` does not handle all errors. Errors that `catch Error(string memory reason)` does not catch include: Underflow/Overflow, Out of Gas, Custom errors, Failing `assert()`, ABI decode errors and `require` without a reason.

Recommendation: Do not specify what type of error can be handled; handle all errors using a generic `catch {}`.

Resolution and Client comment: Acknowledged.

6.2.3 Burning of tokens can block the user from claiming

Severity: *Low*

Context: MemeTokenRaise.sol#442

Description: After adding liquidity, the owner can burn the remaining tokens from the `MemeTokenRaise` contract. The owner may burn the remaining tokens even if there is still a user who has not claimed their tokens. Every user should wait at least `lastClaimedTimestamp + claimLockOffset + userLockOffset`.

Recommendation: Add a requirement for the owner to wait at least `lastClaimedTimestamp + claimLockOffset + (10 to 30 days)` before burning the remaining tokens.

Resolution and Client comment: Acknowledged.