QuillAudits

Audit Report
June, 2024

For

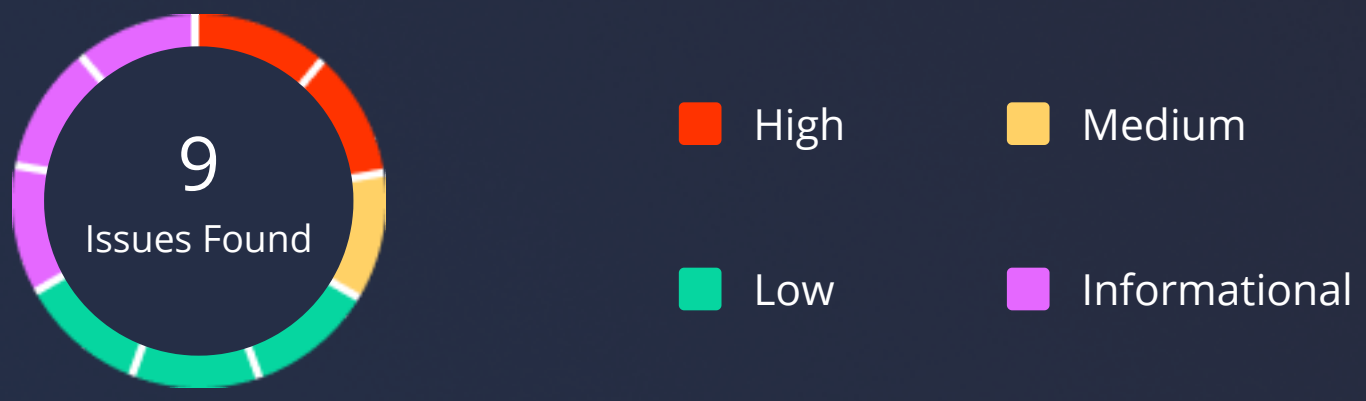
Table of Content

Table of Content

Executive Summary

Project Name	FamilyMetaToken
Overview	FamilyMetaToken is Erc20 Token Contract with upgradeability and pausable functionalities
Timeline	3rd June 2024 to 7th June 2024
Update Code Received	11th June 2024
Second Review	12th June 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the Family Meta Token Contract for quality, security, and correctness.
Source Code	*https://github.com/FamilyMeta-FAMA/FAMA*
Contracts In-Scope	FamilyMeta.sol
Branch	Main
Fixed In	https://github.com/FamilyMeta-FAMA/FAMA/commit/3358196adf85d1c0663147aef115443796fde3ca

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	1	3	3

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC's standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four severity levels, each of which has been explained below.

High Severity Issues

A high severity issue or vulnerability means your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These issues were identified in the initial audit and successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Failing changeOwnership function

Path

FamilyMetaToken

Function

changeOwnership

Description

changeOwnership function is only callable by multisigWallet address to transfer the ownership to the new owner. This function utilizes Openzeppelin's transferOwnership function but it is only callable by the current owner not by the token contract. Therefore ownership is not changed.

```
/**
* @dev Changes the ownership to a new address. Callable by multisig wallet.  * @param newOwner Address of the new owner.
*/
function changeOwnership(address newOwner) external onlyMultisig nonReentrant {
    require(newOwner != address(0), "New owner cannot be zero address.");
    transferOwnership(newOwner); // @audit only callable by the current owner
}
```

POC

Below is the Foundry test which indicates that the call fails

```
function test_change_owner_error() public {
    address newOwner = makeAddr("newOwner");
    vm.prank(multisigWallet, multisigWallet);
    vm.expectRevert();
    familyMeta.changeOwnership(newOwner);
    // ownership is not transferred
    assertNotEq(newOwner, familyMeta.owner());
}
```

Recommendation

Consider to use the transferOwnership function directly (should be called by current owner).

Status

Resolved

QuillAudits' Team Comment

FamilyMetaToken Team replaced the single-step changeOwnership function with a two-step ownership transfer mechanism using requestOwnershipTransfer and confirmOwnershipTransfer. Consequently, the transferOwnership function has been disabled to enforce the two-step process.

2. owner/multisigWallet can burn tokens while the contract is paused

Path

FamilyMetaToken

Function

Burn

Description

The FamilyMetaToken contract can be paused and unpaused by the multisigWallet. burn function is not using the whenNotPaused modifier. When the FamilyMetaToken contract is paused, owner/multisigWallet address can burn user's FAMA tokens. Through this malicious owner can increase the price of the FAMA token.

```
// @audit whenNotPaused modifier is not used
function burn(address from, uint256 amount) external nonReentrant {
    require(amount > 0, "Burn amount must be greater than zero");
    require(msg.sender == multisigWallet || msg.sender == owner(), "Not authorized to
burn");
    _burn(from, amount);
    emit Burned(from, amount);
}
```

Owner can also change ownership or authorize upgrade to new implementation while contract is paused.

Recommendation

Consider to add whenNotPaused modifier to the burn, _authorizeUpgrade, changeOwnership and changeMultisigWallet function.

Status

Resolved

QuillAudits' Team Comment

FamilyMetaToken added whenNotPaused modifier.

Medium Severity Issues

3. ERC20 functionalities not paused/unpaused

Path

FamilyMetaToken

Description

According to the FamilyMeta team, "Our intention in using PausableUpgradeable is indeed to allow us to pause and unpause ERC20 functionalities and other features". PausableUpgradeable's modifier (whenNotPaused) are not used to pause the ERC20 functionalities.

POC

Below is the Foundry test as a PoC

```
function test_transfer_while_paused() public {
    address Alice = makeAddr("Alice");
    address bob = makeAddr("bob");
    vm.prank(owner, owner);
    // owner transfer 1000 FAMA to Alice
    familyMeta.transfer(Alice, 1000);

    vm.prank(multisigWallet, multisigWallet);
    // multisigWallet pauses the token functionalities
    familyMeta.pause();

    vm.prank(Alice, alice);
    // Alice can still transfer the tokens to bob
    familyMeta.transfer(bob, 1000);
    assertEq(familyMeta.balanceOf(Alice), 0);
    assertEq(familyMeta.balanceOf(bob), 1000);
}
```

Recommendation

Consider adding the whenNotPaused modifier to ERC20 functions (burn, transfer, allowance and other) by overriding them to prevent this from occurring while the contract is paused. Another way to solve this to use ERC20PausableUpgradeable instead of PausableUpgradeable and ERC20Upgradeable.

Status

Resolved

QuillAudits' Team Comment

FamilyMetaToken team overridden the transfer, allowance, transferFrom, and approve functions to include the whenNotPaused modifier.

Low Severity Issues

4. Use Ownable2StepUpgradeable instead of OwnableUpgradeable

Path

FamilyMetaToken

Description

Token contract uses one-step ownership transfer through OwnableUpgradeable.

Two-step ownership transfer is preferable, where:

- The current owner proposes a new address for the ownership change.
- In a separate transaction, the proposed new address can then claim the ownership.

Recommendation

Consider doing two step owner transfer to be safe and secure using Ownable2StepUpgradeable

Status

Resolved

QuillAudits' Team Comment

FamilyMetaToken team implemented two-step ownership transfer.

5. Mismatch of decimals value

Path

FamilyMetaToken

Description

Based on the whitepaper, decimals for FAMA token is 8. But the token contract is using 18 decimals (Openzeppelin ERC20 tokens use this).

Recommendation

Consider refactoring the token contract/whitepaper.

Status

Resolved

QuillAudits' Team Comment

FamilyMetaToken team changed the whitepaper.

6. Wrong Openzeppelin dependencies import path

Path

FamilyMetaToken

Description

Openzeppelin imports (ReentrancyGuardUpgradeable and PausableUpgradeable) in the contract use old paths. Changing the paths based on the new version of Openzeppelin is better otherwise there will be a compilation error.

Recommendation

Consider changing paths of the imports in the contract.

Status

Resolved

Informational Issues

7. Already existing decimals function

Path

FamilyMetaToken

Function

decimal

Description

decimals function in the contract overrides the openzeppelin's decimals function but logic is not changed. Token is initialized with 18 decimals by default which is returned by both versions of the function.

```
/ @audit overriding the function is not changing the logic
function decimals() public pure override returns (uint8) {
    return _decimals;
}
```

Recommendation

Consider to use the existing Openzeppelin's decimals function for better readability. Also reduces the contract size and complexity.

Status

Resolved

QuillAudits' Team Comment

FamilyMetaToken team removed the override of decimals function and used the default OpenZeppelin implementation.

8. Use underscores as separators

Path

FamilyMetaToken

Description

In Solidity, you can use underscores as separators within numeric literals to make them easier to read. For example, 50000000000 can be made into 50_000_000_000.

Recommendation

Consider to use underscores for better readability.

Status

Resolved

QuillAudits Team's Comment

FamilyMetaToken team applied underscores to large numeric literals for better readability, e.g., 50_000_000_000.

9. Use owner instead of multisigWallet

Description

Based on the FamilyMeta team "for management optimization, we internally decided to treat the owner and the multisig wallet as the same". Contract uses both of these addresses.

Recommendation

Consider to use just owner throughout the contract and remove multisigWallet.

Status

Resolved

QuillAudits Team's Comment

FamilyMetaToken team is using owner throughout the contract.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the FamilyMeta Contract. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, the Family Meta Team Resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in FamilyMeta smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of FamilyMeta smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the FamilyMeta to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com