0xCommit

Security Assessment Report

FAMProtocol

FAMProtocol contracts Audit

Version: Final

Date: 4 Oct 2024

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

Introduction

Purpose of this report

0xCommit has been engaged by **FAMProtocol** to perform a security audit of several bridge contract components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following contracts:

https://github.com/Metageeks-Technologies/FAMProtocol-SmartContracts/blob/Vivek_dev/contracts/

Description	Commit hash
Initial Commit	bfe61a151d45a4ff3c210517ac3507ffdbd658c8
Final Commit	07eaed821e4ef149f525d7b6a137780d2a9609a1

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
High	An attacker can successfully execute an attack that clearly results in operational issues for the service. This also includes any value loss of unclaimed funds permanently or temporary.
Medium	The service may be susceptible to an attacker carrying out an unintentional action, which could potentially disrupt its operation. Nonetheless, certain limitations exist that make it difficult for the attack to be successful.
Low	The service may be vulnerable to an attacker executing an unintended action, but the impact of the action is negligible or the likelihood of the attack succeeding is very low and there is no loss of value.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
    d. Access Control Issues
    e. Boundary Analysis
4. Report preparation

Summary of Findings

Sr. No.	Description	Severity	Status
1	Exploitation of Free Mint Referrals	High	Resolved
2	ERC20 tokens with no return value will fail to transfer	Medium	Resolved
3	Hardcodes decimals in DOMAIN_MINT_FEE	Low	Resolved
4	Missing Event Emissions for State Changes	Informational	Resolved

Detailed Findings

Critical Findings

No Critical findings found.

High Findings

1. Exploitation of Free Mint Referrals

Severity: High

Description

The function `createReferralCode` allows the contract owner to add new referral codes. Since these codes are stored on-chain and visible to all users, an attacker could exploit this by running a script to claim all available free mints using multiple addresses. This would prevent legitimate users from being able to claim their free mint.

Additionally, the attacker could exploit this on discounted mints as well, though they would need to provide the required funds for the mint fee in that case.

Impact

- Legitimate users are denied access to free mints.
- Potential abuse of discounted mints, although this requires the attacker to pay the fee.

Remediation

Consider implementing an off-chain mechanism to store referral codes or limit the visibility of referral codes in a way that prevents attackers from exploiting the system.

Status

Resolved

FAM Protocol team removed the referral code logic for these functions, and only the whitelisted addresses can mint the domain

Medium Findings

2. ERC20 tokens with no return value will fail to transfer

Severity: Medium

Description

Although the ERC20 standard suggests that a transfer should return true on success, many tokens are non-compliant in this regard (including high profile, like USDT) . In that case, the `.transfer()` call here will revert even if the transfer is successful, because solidity will check that the RETURNDATASIZE matches the ERC20 interface.

Remediation

Consider using OpenZeppelin's SafeERC20 library functions.

Status

Resolved

OpenZeppelin's SafeERC20 library functions were implemented.

Low Findings

3. Hardcodes decimals in DOMAIN_MINT_FEE

Severity: Low

Description

The `DOMAIN_MINT_FEE` is hardcoded to use 6 decimal places, assuming that the `paymentToken` will always be USDT, which has 6 decimals. However, the `paymentToken` can be updated by the owner post-deployment to a token with a different decimal precision. If the paymentToken has more or fewer than 6 decimals, this could lead to incorrect fee calculations.

Impact

- **Incorrect Fee Calculation:** If a token with a different decimal precision is used, the DOMAIN_MINT_FEE may be too high or too low.
- **Transaction Reversions:** In some cases, incorrect fee calculations may lead to transaction failures, resulting in reverted transactions.

Remediation

Instead of hardcoding the decimal value, calculate the DOMAIN_MINT_FEE dynamically based on the paymentToken's decimals. This can be done by fetching the decimals() value from the token contract after the paymentToken is set.

Status

Resolved

The Protocol team has mentioned they will only use USDT as the payment token and make sure it has 6 decimals on the chain they deploy. They function to update the payment token was also removed.

Informational Findings

4. Missing Event Emissions for State Changes

Severity: Informational

Description

The contract does not emit events for important state changes, which makes it difficult to track and audit contract behavior off-chain.

Remediation

Ensure that events are emitted for all important state changes. Use meaningful names for events and include relevant parameters to help external systems and users track contract activity effectively.

Status

Resolved