SAFETIN

AUDIT

Frak

November 7th, 2022

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for Frak in order to find flaws and vulnerabilities in the Frak project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and Frak Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	Frak
Description	The Frak Ecosystem is built to align interests of Creators and their Community in order to share the value of the content more fairly, that revolutionize the monetization of culture.
Platform	Polygon
Language	Solidity
Codebase	FrakTokenL2.sol

FINDINGS SUMMARY

Vulnerability	Total	Resolved
Critical	0	0
Major	1	0
Medium	3	0
Minor	2	0
Informational	4	0

AUDIT FINDINGS

Code	Title	Severity
EXT-1	Depends to an external protocol	Minor
COMP-1	Unfixed version of compiler	Minor
SUPPL-1	Checking cap when depositing	Major
HOOK-1	Specify WhenNotPaused on the entry-level	Medium
MINT-2	Improve Consistency in usage of WhenNotPaused	Medium
VAR-1	The cap state variable can be initialised as a constant variable	Informational
VAR-2	Fetch the cap from the constant variable	Informational
OPCODE-1	Save gas checking less than instead of less than or equal	Informational

	in mint function	
UINT-1	Unusual usage of decoding to access uint	Informational
PAUSE-1	it will impact the ability for tokens being bridged across chains	Medium

EXT-1 | Dependence to an external protocol

Description

The contract interacts with the third party Polygon Bridge mechanism. The scope of the audit would treat this third party entity as a black box and assume it is fully functional. However, this protocol contains several entities that should be verified before they are assigned roles. For example the DEPOSITOR_ROLE.

Recommendation

Monitor the functionality of the bridge and Validate that the correct addresses are being assigned to the used roles: DEPOSITOR_ROLE.

COMP-1 | Unfixed version of compiler

Description

contract does not have locked compiler versions, meaning a range of compiler versions can be used. This can lead to differing bytecodes being produced depending on the compiler version, which can create confusion when debugging as bugs may be specific to a specific compiler version(s).

Recommendation

To rectify this, we recommend setting the compiler to a single version, the version tested the most to be compatible with the code, an example of this change can be seen below.

```
pragma solidity 0.8.7;
```

SUPPL-1 | Checking cap when depositing

Description

There is no check when calling deposit that the new tokens being minted should be less than the totalSupply plus the mint amount. If a restriction should be imposed on both minting functions then this require check should be handled before the mint function is called.

Recommendation

Should add the following as the first line of code in deposit() function

require(totalSupply() + amount < cap, "CAP_REACHED");

HOOK-1 | Specify WhenNotPaused on the entry-level

Description

WhenNotPaused is used on the beforeTokenTransfer hook, which will be checked on all mint, burn, and transfer functions for an ERC20 token. If the purpose of the pause check is to limit these functions under specific conditions then use whenNotPaused on the highest level i.e. when mint, burn, or transfer are being called.

WhenNotPaused is used on the deposit function to prevent tokens from being minted if the contract is paused. However, it will be checked twice as it is called again in the beforeTokenTransfer hook when the mint action is called.

Recommendation

Use whenNotPaused on external/ public function level rather than on the hook

MINT-2 | Improve Consistency in usage of WhenNotPaused

Description

WhenNotPaused is used to prevent minting of new tokens via deposit but minting remains available through the mint function. If the use of paused is to prevent adversarial minting of tokens then protecting all entry points to minting makes sense.

Recommendation

Add whenNotPaused check to mint function

VAR-1 | The cap state variable can be initialised as a constant variable

Description

The cap variable being used to limit supply remains constant for the life of the contract and this means it can be set as a constant variable and does not need to be initialised through the initialize function.

Recommendation

Add the cap in supply to the state variable declaration

```
uint256 private constant _cap = 3_000_000_000 ether
```

VAR-2 | Fetch the cap from the constant variable

Description

When fetching the value for _cap, a call to cap() is made which can be simplified to directly fetching the data from the constant variable _cap.

Recommendation

Fetch the cap from the constant variable

require(totalSupply() + amount < _cap, “CAP_EXCEEDED”);

OPCODE-1 | Save gas checking less than instead of less than or equal in mint function

Description

Checking less than in solidity uses a single opcode whereas checking less than or equal to uses the LT/GT opcode and afterwards it executes an ISZERO opcode to check the result of the previous comparison is zero. If saving gas is important then changing this check to a less-than check will help.

Recommendation

Change check to less than

require(totalSupply() + amount < _cap, “CAP_EXCEEDED”);

UINT-1 | Unusual usage of decoding to access uint

Description

In the deposit function, a bytes parameter is passed in and the value is decoded into a uint before being used to mint. This is an unusual pattern unless there is cross-contract communication where the input value will be of bytes type or for another specific reason. If this is not the case, pass the uint in as a uint256 input directly to the function.

Recommendation

Pass the uint256 in as a function input function deposit(address user, bytes calldata depositData) external whenNotPaused

PAUSE-1 | It will impact the ability for tokens being bridged across chains

Description

If the contract has been paused it will impact the ability for tokens being bridged across chains. As the bridge normally takes 15-30 mins for settle transactions, there are examples where tokens could be in the process of being bridged and when the validators try to call deposit to complete the bridge it will not be possible.

Recommendation

Remove the pause functionality from the deposit and beforeTokenTransfer function to resolve bridging issue. Specifically, add pause to the functions they do not want to be called in the case they need to pause.

Global security warnings

These are safety issues for the whole project. They are not necessarily critical problems but they are inherent in the structure of the project itself. Potential attack vectors for these security problems should be monitored.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without Safetin's prior written consent.This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Safetin to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or

legal compliance. This report should not be used in any way

Safetin security assessment to make decisions around investment or involvement with any particular project.

This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Safetin's position is that each company and individual are responsible for their own due diligence and continuous security. Safetin's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or fun.