Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Frax Ether
Platform: Ethereum
Language: Solidity
Date: January 27th, 2024

Table of contents

Introduction 4

Project Background 4

Audit Scope 5

Claimed Smart Contract Features 6

Audit Summary 7

Technical Quick Stats 8

Business Risk Analysis 9

Code Quality 10

Documentation 10

Use of Dependencies 10

AS-IS overview 11

Severity Definitions 12

Audit Findings 13

Conclusion 16

Our Methodology 17

Disclaimers 19

Appendix

- Code Flow Diagram 20
- Slither Results Log 21
- Solidity static analysis 23
- Solhint Linter 25

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Frax Ether Token smart contract from app.frax.finance was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on January 27th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The Frax Ether is an ERC20-based smart contract. Here's an overview of the key functionalities added:
- **Ownership and Governance:** The Owned contract allows for ownership control and governance, including the ability to nominate and accept new owners.
- **Minter Permissions:**
    - The contract includes an array and a mapping to manage a list of minters who have the ability to mint new tokens.
    - Functions to add and remove minters with appropriate access control using onlyByOwnGov modifier.
- **Minting and Burning:**
    - Minters can mint new tokens and burn tokens from a specified address using the minter_mint and minter_burn_from functions respectively.
    - Events are emitted to log the minting and burning activities.
- **Timelock Control:** The contract includes functionality to set and change a timelock address which can also perform governance functions alongside the owner.
- **Access Control Modifiers:**
    - onlyByOwnGov ensures that only the owner or timelock address can perform certain actions.
    - onlyMinters ensures that only approved minters can mint or burn tokens.
- These additions ensure that the contract has robust access control and governance mechanisms while allowing for flexible minting and burning of tokens by authorized addresses.

Audit scope

Name	Code Review and Security Analysis Report for Frax Ether Smart Contract
Platform	Ethereum
Language	Solidity
File	frxETH.sol
Ethereum Code	0x5e8422345238f34275888049021821e8e08caa1f
Audit Date	January 27th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Frax Ether • Symbol: frxETH • Decimals: 18	**YES, This is valid.**
Ownership Control: • Update whitelisted minters addressed by the owner. • The owner can set a timelock. • The current owner can nominate ownership of the contract to a new account. • Nominate owners can accept ownership.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. This token contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 3 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version not specified	Passed
	Solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	No
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	Yes
Can Mint?	No
Is it Proxy?	No
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Frax Ether are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Frax Ether.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Frax Ether smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyByOwnGov	modifier	Passed	No Issue
3	onlyMinters	modifier	Unlimited minting	Refer Audit Findings
4	minter_burn_from	write	naming-convention, Centralization, Gas Optimization	Refer Audit Findings
5	minter_mint	write	naming-convention, Centralization, Gas Optimization	Refer Audit Findings
6	addMinter	write	Centralization, Gas Optimization	Refer Audit Findings
7	removeMinter	write	Centralization, Gas Optimization	Refer Audit Findings
8	setTimelock	write	Gas Optimization	Refer Audit Findings
9	nominateNewOwner	external	Centralization	Refer Audit Findings
10	acceptOwnership	external	Passed	No Issue
11	onlyOwner	modifier	Passed	No Issue
12	burn	write	Passed	No Issue
13	burnFrom	write	Passed	No Issue
14	permit	write	Passed	No Issue
15	nonces	read	Passed	No Issue
16	DOMAIN_SEPARATOR	external	Passed	No Issue
17	_useNonce	internal	Passed	No Issue
18	_domainSeparatorV4	internal	Passed	No Issue
19	_buildDomainSeparator	read	Passed	No Issue
20	_hashTypedDataV4	internal	Passed	No Issue
21	_msgSender	internal	Passed	No Issue
22	_msgData	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Unlimited minting:
The OnlyMinter can mint unlimited tokens in this contract.

Resolution: We suggest setting the max supply and using the required conditions in the mint function and check the max supply.

Very Low / Informational / Best practices:

(1) naming-convention:

```
// Used by minters when user redeems
function minter_burn_from(address b_address, uint256 b_amount) public onlyMinters {
    super.burnFrom(b_address, b_amount);
    emit TokenMinterBurned(b_address, msg.sender, b_amount);
}

// This function is what other minters will call to mint new tokens
function minter_mint(address m_address, uint256 m_amount) public onlyMinters {
    super._mint(m_address, m_amount);
    emit TokenMinterMinted(msg.sender, m_address, m_amount);
}
```

Solidity defines a naming convention that should be followed.

Resolution: Follow the Solidity naming convention.

(2) Centralization:

OnlyMinters can mint unlimited tokens and OnlyMinters has owner authority of these functions:

- minter_burn_from()
- minter_mint()

onlyOwner has an owner authority:

- nominateNewOwner()

onlyByOwnGov has owner authority on:

- addMinter()
- removeMinter()

Any compromise to these accounts may allow the hacker to manipulate the project through these functions.

Resolution: We suggest carefully managing these account's private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

(3) Gas Optimization:

Public functions that are never called by the contract could be declared external.

- setTimelock()
- removeMinter()
- addMinter()
- minter_mint()
- minter_burn_from()

Resolution: We suggest declaring these all functions external for better Gas optimization.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

ERC20PermitPermissionedMint.sol

- addMinter: Add whitelisted minters addressed by the owner.
- removeMinter: Remove a minter addressed by the owner.
- setTimelock: The owner can set a timelock.

Owned.sol

- nominateNewOwner: The current owner can nominate ownership of the contract to a new account.
- acceptOwnership: Nominate owner can accept ownership.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 low and 3 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

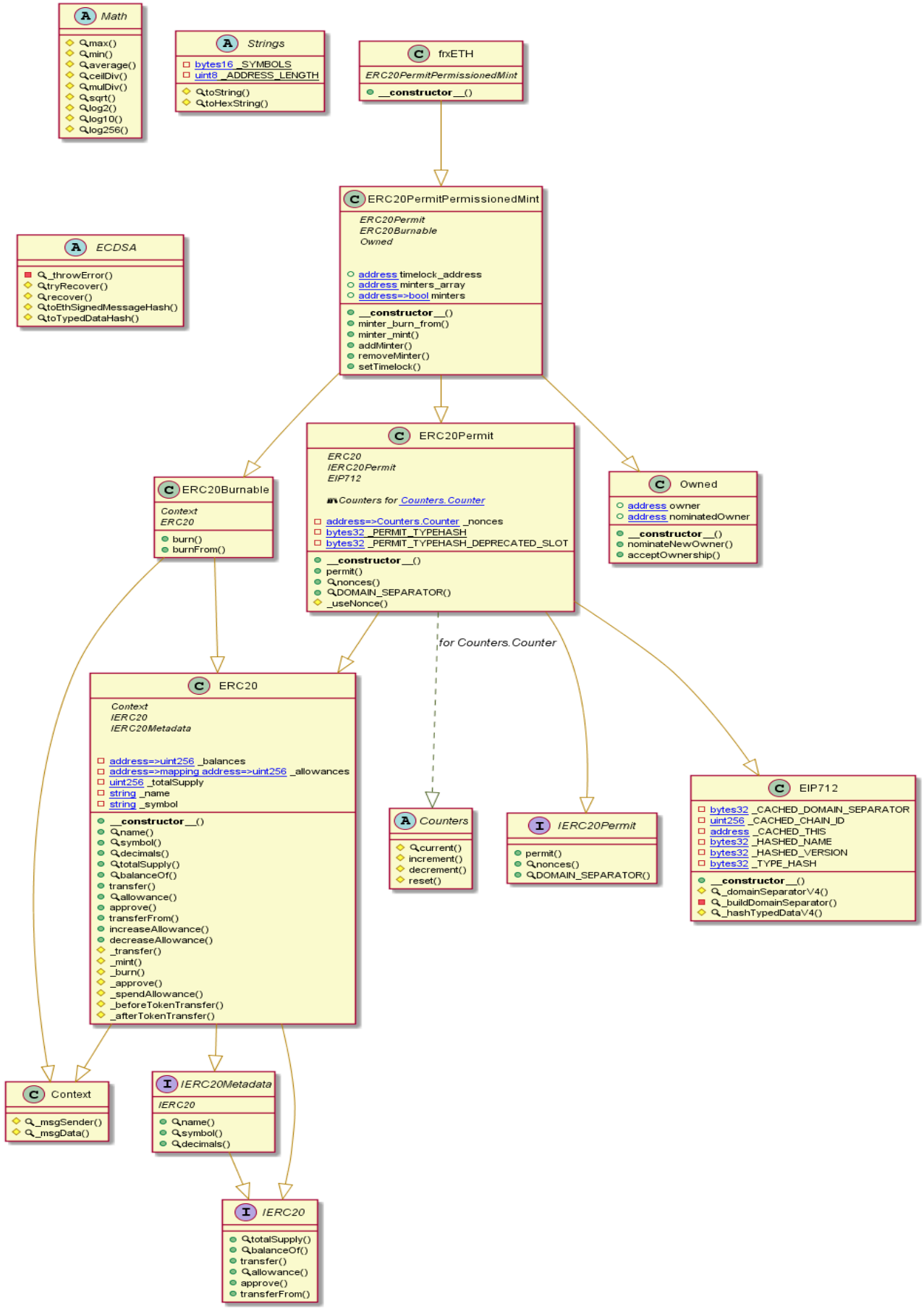
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Frax Ether



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> frxETH.sol

```
INFO:Detectors:
ERC20Permit.constructor(string).name (frxETH.sol#1373) shadows:
        - ERC20.name() (frxETH.sol#195-197) (function)
        - IERC20Metadata.name() (frxETH.sol#108) (function)
ERC20PermitPermissionedMint.constructor(address,address,string,string)._name (frxETH.sol#1510) shadows:
        - ERC20._name (frxETH.sol#175) (state variable)
ERC20PermitPermissionedMint.constructor(address,address,string,string)._symbol (frxETH.sol#1511) shadows:
        - ERC20._symbol (frxETH.sol#176) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ERC20PermitPermissionedMint.constructor(address,address,string,string)._timelock_address (frxETH.sol#1509) lacks a zero-check on :
                - timelock_address = _timelock_address (frxETH.sol#1517)
Owned.nominateNewOwner(address)._owner (frxETH.sol#1472) lacks a zero-check on :
                - nominatedOwner = _owner (frxETH.sol#1473)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (frxETH.sol#1378-1397) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (frxETH.sol#1387)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (frxETH.sol#639-719) uses assembly
        - INLINE ASM (frxETH.sol#650-654)
```

```
        - INLINE ASM (frxETH.sol#670-677)
        - INLINE ASM (frxETH.sol#684-693)
Strings.toString(uint256) (frxETH.sol#941-961) uses assembly
        - INLINE ASM (frxETH.sol#947-949)
        - INLINE ASM (frxETH.sol#953-955)
ECDSA.tryRecover(bytes32,bytes) (frxETH.sol#1042-1059) uses assembly
        - INLINE ASM (frxETH.sol#1050-1054)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
ERC20PermitPermissionedMint.addMinter(address) (frxETH.sol#1547-1555) compares to a boolean constant:
        -require(bool,string)(minters[minter_address] == false,Address already exists) (frxETH.sol#1550)
ERC20PermitPermissionedMint.removeMinter(address) (frxETH.sol#1558-1574) compares to a boolean constant:
        -require(bool,string)(minters[minter_address] == true,Address nonexistant) (frxETH.sol#1560)
ERC20PermitPermissionedMint.onlyMinters() (frxETH.sol#1527-1530) compares to a boolean constant:
        -require(bool,string)(minters[msg.sender] == true,Only minters) (frxETH.sol#1528)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (frxETH.sol#138-140) is never used and should be removed
Counters.decrement(Counters.Counter) (frxETH.sol#1328-1334) is never used and should be removed
Counters.reset(Counters.Counter) (frxETH.sol#1336-1338) is never used and should be removed
ECDSA.recover(bytes32,bytes) (frxETH.sol#1075-1079) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (frxETH.sol#1103-1111) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes) (frxETH.sol#1184-1186) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (frxETH.sol#1170-1174) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes) (frxETH.sol#1042-1059) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (frxETH.sol#1088-1096) is never used and should be removed
```

```
Math.sqrt(uint256) (frxETH.sol#742-773) is never used and should be removed
Math.sqrt(uint256,Math.Rounding) (frxETH.sol#778-783) is never used and should be removed
Strings.toHexString(address) (frxETH.sol#990-992) is never used and should be removed
Strings.toHexString(uint256) (frxETH.sol#966-970) is never used and should be removed
Strings.toHexString(uint256,uint256) (frxETH.sol#975-985) is never used and should be removed
Strings.toString(uint256) (frxETH.sol#941-961) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.8.0 (frxETH.sol#6) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IERC20Permit.DOMAIN_SEPARATOR() (frxETH.sol#581) is not in mixedCase
Variable EIP712._CACHED_DOMAIN_SEPARATOR (frxETH.sol#1225) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (frxETH.sol#1226) is not in mixedCase
Variable EIP712._CACHED_THIS (frxETH.sol#1227) is not in mixedCase
Variable EIP712._HASHED_NAME (frxETH.sol#1229) is not in mixedCase
Variable EIP712._HASHED_VERSION (frxETH.sol#1230) is not in mixedCase
Variable EIP712._TYPE_HASH (frxETH.sol#1231) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (frxETH.sol#1410-1412) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (frxETH.sol#1366) is not in mixedCase
Parameter Owned.nominateNewOwner(address)._owner (frxETH.sol#1472) is not in mixedCase
Function ERC20PermitPermissionedMint.minter_burn_from(address,uint256) (frxETH.sol#1535-1538) is not in mixedCase
```

```
Function ERC20PermitPermissionedMint.minter_burn_from(address,uint256) (frxETH.sol#1535-1538) is not in mixedCase
Parameter ERC20PermitPermissionedMint.minter_burn_from(address,uint256).b_address (frxETH.sol#1535) is not in mixedCase
Parameter ERC20PermitPermissionedMint.minter_burn_from(address,uint256).b_amount (frxETH.sol#1535) is not in mixedCase
Function ERC20PermitPermissionedMint.minter_mint(address,uint256) (frxETH.sol#1541-1544) is not in mixedCase
Parameter ERC20PermitPermissionedMint.minter_mint(address,uint256).m_address (frxETH.sol#1541) is not in mixedCase
Parameter ERC20PermitPermissionedMint.minter_mint(address,uint256).m_amount (frxETH.sol#1541) is not in mixedCase
Parameter ERC20PermitPermissionedMint.addMinter(address).minter_address (frxETH.sol#1547) is not in mixedCase
Parameter ERC20PermitPermissionedMint.removeMinter(address).minter_address (frxETH.sol#1558) is not in mixedCase
Parameter ERC20PermitPermissionedMint.setTimelock(address)._timelock_address (frxETH.sol#1576) is not in mixedCase
Variable ERC20PermitPermissionedMint.timelock_address (frxETH.sol#1499) is not in mixedCase
Variable ERC20PermitPermissionedMint.minters_array (frxETH.sol#1502) is not in mixedCase
Contract frxETH (frxETH.sol#1591-1602) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Loop condition i < minters_array.length (frxETH.sol#1566) should use cached array length instead of referencing `length` member of the storage
 array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Slither:frxETH.sol analyzed (15 contracts with 93 detectors), 73 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

frxETH.sol

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 1072:16:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 1409:23:

Gas costs:

Gas requirement of function frxETH.permit is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1400:11:

Email: audit@EtherAuthority.io

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 1588:15:

Constant/View/Pure functions:

ERC20Permit._useNonce(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1441:11:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1550:14:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

more

Pos: 1585:15:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

frxETH.sol

```
Compiler version >=0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:5
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:208
Error message for require is too long
Pos: 9:358
Error message for require is too long
Pos: 9:385
Error message for require is too long
Pos: 9:445
Error message for require is too long
Pos: 9:475
Error message for require is too long
Pos: 9:476
Code contains empty blocks
Pos: 24:522
Code contains empty blocks
Pos: 24:542
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:671
Provide an error message for require
Pos: 13:683
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:691
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:705
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:968
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:974
Error message for revert is too long
Pos: 13:1039
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:1071
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1268
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1394
Code contains empty blocks
Pos: 55:1394
Avoid making time-based decisions in your business logic
```

```
Pos: 17:1408
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1487
Error message for require is too long
Pos: 9:1499
Error message for require is too long
Pos: 9:1506
Variable name must be in mixedCase
Pos: 5:1520
Variable name must be in mixedCase
Pos: 5:1523
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1528
Variable name must be in mixedCase
Pos: 9:1529
Variable name must be in mixedCase
Pos: 9:1530
Function name must be in mixedCase
Pos: 5:1556
Variable name must be in mixedCase
Pos: 31:1556
Variable name must be in mixedCase
Pos: 50:1556
Function name must be in mixedCase
Pos: 5:1562
Variable name must be in mixedCase
Pos: 26:1562
Variable name must be in mixedCase
Pos: 45:1562
Variable name must be in mixedCase
Pos: 24:1568
Variable name must be in mixedCase
Pos: 23:1607
Variable name must be in mixedCase
Pos: 25:1608
Variable name must be in mixedCase
Pos: 27:1609
Contract name must be in CamelCase
Pos: 1:1612
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1615
Variable name must be in mixedCase
Pos: 7:1616
Variable name must be in mixedCase
Pos: 7:1617
Code contains empty blocks
Pos: 5:1620
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io