Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: SleeFi Protocol
Website: https://sleefi.com/en/
Platform: Avalanche Network
Language: Solidity
Date: June 21st, 2022

Table of contents

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

EtherAuthority was contracted by the SleeFi team to perform the Security audit of the smart contracts code. 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 June 21st, 2022.

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

SleeFi protocol is using SLGT as a governance token, SLFT as a utility token and vesting protocol is for VCs. SleeFi contract inherits the OwnableUpgradeable, SafeERC20Upgradeable, IERC20Upgradeable, ERC20Upgradeable, Initializable, ERC20BurnableUpgradeable standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for SleeFi Protocol Smart Contracts**
Platform	**Avalanche / Solidity**
File 1	SLFTToken.sol
File 1 MD5 Hash	BC56A982DE3B231839DE84A59B1B4152
File 2	SLGTToken.sol
File 2 MD5 Hash	2E8BFE0EBC85FAEC009909E03A656058
File 3	SleeFiVestingVault.sol
File 3 MD5 Hash	FF8E08965425A93ABF5C7D3DAD671C5C
Audit Date	June 21st, 2022
Revision Date	June 23rd, 2022

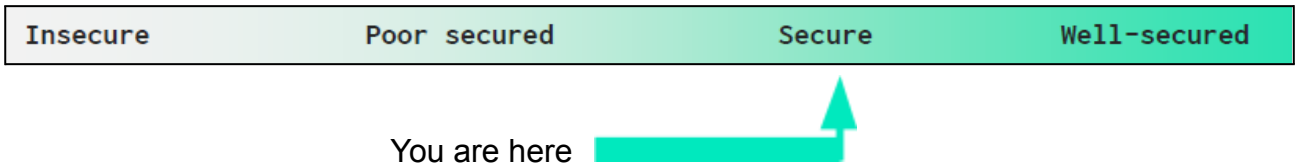
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 SLFTToken.sol • Name: SLFT • Symbol: SLFT	**YES, This is valid.**
File 2 SLGTToken.sol • Name: SLGT • Symbol: SLGT • Total Supply: 12 billion	**YES, This is valid.**
File 3 SleeFiVestingVault.sol • SleeFiVestingVault can apply Vesting. • SleeFiVestingVault owner can update Vesting Schedule.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 1 medium and 3 low and some very low level issues.
All the issues have been acknowledged.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	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	Acknowledged
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 3 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the SleeFi Protocol 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 SleeFi Protocol.

The SleeFi team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are not well commented on smart contracts. We suggest using Ethereum's NatSpec style for the commenting.

Documentation

We were given a SleeFi Protocol smart contract code in the form of a snowtrace.io web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented. So it is not 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.

Another source of information was its official website https://sleefi.com/en/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

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

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

SLFTToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	__Ownable_init	internal	access only Initializing	No Issue
7	__Ownable_init_unchained	internal	access only Initializing	No Issue
8	onlyOwner	modifier	Passed	No Issue
9	owner	read	Passed	No Issue
10	_checkOwner	internal	Passed	No Issue
11	renounceOwnership	write	access only Owner	No Issue
12	transferOwnership	write	access only Owner	No Issue
13	_transferOwnership	internal	Passed	No Issue
14	__ERC20Burnable_init	internal	access only Initializing	No Issue
15	__ERC20Burnable_init_unchained	internal	access only Initializing	No Issue
16	burn	write	Passed	No Issue
17	burnFrom	write	Passed	No Issue
18	initialize	external	access by initializer	No Issue
19	_mint	internal	No Max minting of the tokens set	Acknowledged
20	_afterTokenTransfer	internal	Empty function used	Acknowledged
21	_beforeTokenTransfer	internal	Empty function used	Acknowledged

SLGTToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	__Ownable_init	internal	access only Initializing	No Issue
7	__Ownable_init_unchained	internal	access only Initializing	No Issue

8	onlyOwner	modifier	Passed	No Issue
9	owner	read	Passed	No Issue
10	_checkOwner	internal	Passed	No Issue
11	renounceOwnership	write	access only Owner	No Issue
12	transferOwnership	write	access only Owner	No Issue
13	_transferOwnership	internal	Passed	No Issue
14	initialize	external	access by initializer	No Issue
15	_afterTokenTransfer	internal	Empty function used	Acknowledged
16	_beforeTokenTransfer	internal	Empty function used	Acknowledged

SleeFiVestingVault.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	__Ownable_init	internal	access only Initializing	No Issue
7	__Ownable_init_unchained	internal	access only Initializing	No Issue
8	onlyOwner	modifier	Passed	No Issue
9	owner	read	Passed	No Issue
10	_checkOwner	internal	Passed	No Issue
11	renounceOwnership	write	access only Owner	No Issue
12	transferOwnership	write	access only Owner	No Issue
13	_transferOwnership	internal	Passed	No Issue
14	initialize	external	access by initializer	No Issue
15	_getUserVestingInfo	internal	Passed	No Issue
16	getUserVestingInfo	external	The msg.sender used in view function	Acknowledged
17	updateVestingSchedule	external	Function input parameters lack of check	Acknowledged
18	_applyVesting	internal	Passed	No Issue
19	applyVesting	external	access only Owner	No Issue
20	claim	external	Passed	No Issue
21	_effectiveDay	internal	Passed	No Issue
22	_getVestedInfo	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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No critical severity vulnerabilities were found.

High Severity

No high severity vulnerabilities were found.

Medium

(1) No Max minting of the tokens set: **SLFTToken.sol**

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

Setting max minting for the tokens is good for tokenomics.Token minting without any maximum limit is considered inappropriate for tokenomics.

Resolution: Token minting without any maximum limit is considered inappropriate for tokenomics. We recommend placing some limit on token minting to mitigate this issue.

Status: This issue is acknowledged by the SleeFi team as a required business logic as a necessity of the maximum supply because of the game system.

Low

(1) Require can be merge: **SleeFiVestingVault.sol**

```
uint256 grantedAmount = user.amountOfGrant;
require(grantedAmount != 0, "No vesting info");
uint256 claimedAmount = userClaimedAmount[msg.sender][token];
require(claimedAmount < grantedAmount, "Nothing to claim");
VestingSchedule storage userSchedule = _vestingSchedule[msg.sender][
    token
];
require(userSchedule.startTime != 0, "No vesting info");
(, uint256 vestedAmount) = _getVestedInfo(
```

```
VestingInfo storage userInfo = _vestingInfo[beneficiary][token];
require(userInfo.amountOfGrant == 0, "Already applied vesting");
VestingSchedule storage userSchedule = _vestingSchedule[beneficiary][
    token
];
require(userSchedule.startTime == 0, "Already applied vesting");
```

In _applyVesting and Claim functions, 2 requirements with the same messages can be merged to reduce some gas.

Resolution: We suggest merging both require in one.

Status: This issue is acknowledged by the SleeFi team as, It is a required gas to deploy contract will decrease (we just need deploy 1 time only), but the gas to execute these functions will be increase a little bit (these functions can be called many time in future).

(2) The msg.sender used in view function: **SleeFiVestingVault.sol**

```
function getUserVestingInfo(uint256 onDayOrToday)
    external
    view
    returns (VestingSummary[2] memory userVestingData)
{
    userVestingData[0] = _getUserVestingInfo(
        msg.sender,
        SLFT_CONTRACT,
        onDayOrToday
    );
    userVestingData[1] = _getUserVestingInfo(
        msg.sender,
        SLGT_CONTRACT,
        onDayOrToday
    );
```

getUserVestingInfo function is used to fetch the vesting info for the caller so it is using msg.sender. But that does not give vesting info. As there is no other view function to fetch the user's vesting information, it will create problems while showing data at UI.

Resolution: We suggest adding an address parameter and use it instead of msg.sender, it will give vesting info.

Status: This issue is acknowledged by the SleeFi team as being unnecessary for the UI to get the information of a VC at the `Get Vesting Info` screen. The input parameter was removed because It was not desired to get the VC vesting info easily.

(3) Function input parameters lack of check: **SleeFiVestingVault.sol**

```
function updateVestingSchedule(
    address beneficiary,
    address vestingToken,
    uint256 start,
    uint256 cliff,
    uint256 dur,
    uint256 percent
) external onlyOwner {
    VestingSchedule storage userSchedule = _vestingSchedule[beneficiary][
        vestingToken
    ];
    require(userSchedule.startTime > 0 &&
        block.timestamp < userSchedule.startTime,
        "No vesting info or vesting started"
    );
    userSchedule.startTime = start;
    userSchedule.cliffDuration = cliff;
    userSchedule.duration = dur;
    userSchedule.unlockPercent = percent;

    emit VestingScheduleUpdated(
        beneficiary,
        vestingToken,
        start,
        cliff,
        dur,
        percent
    );
}
```

In updateVestingSchedule function owner can set any amount of percent in vesting and there may be misuse of this function.

Resolution: Add a “require” condition in updateVestingSchedule for parent and check if its not > BASE_PRECISION, also not allow more than 100%.

Status: This issue is acknowledged by the SleeFi team as this will be taken care of by the owner when updating the vesting schedule.

Very Low / Informational / Best practices:

(1) Empty function used:

SLFTToken.sol

```
function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual {}
```

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual {}
```

SLGTToken.sol

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual {}

/**
 * @dev Hook that is called after any transfer of tokens. This includes
 * minting and burning.
 *
 * Calling conditions:
 *
 * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
 * has been transferred to `to`.
 * - when `from` is zero, `amount` tokens have been minted for `to`.
 * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
 * - `from` and `to` are never both zero.
 *
 * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
 */
function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual {}
```

The _beforeTokenTransfer and _afterTokenTransfer are hooks and used many times but have no effect on code or anything.

Resolution: This function can be used as a hook.

Status: Acknowledged.

(2) Variable should be immutable: **SleeFiVestingVault.sol**

```
address public SLFT_CONTRACT;
address public SLGT_CONTRACT;

uint256 public constant SECONDS_PER_MONTH = 60 * 60 * 24 * 30;
uint256 public BASE_PRECISION;
```

Variables that are defined within the constructor, but further remain unchanged should be marked as immutable to save gas and to ease the reviewing process of third-parties.

Resolution: Consider marking this variable as immutable.

Status: Acknowledged.

Centralization

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

- updateVestingSchedule: SleeFiVestingVault owner can update vesting schedule.
- applyVesting: SleeFiVestingVault owner can apply vesting.
- mint: SLFTToken owner can mint a token.

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 snowtrace.io web link. And we have used all possible tests based on given objects as files. We have not observed any major issues in the smart contracts. **So, the smart contracts are ready for the mainnet deployment.**

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 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 contract, based on standard audit procedure scope, is **"Secured"**.

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 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 whitebox penetration testing. We look at the project's web site 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, 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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

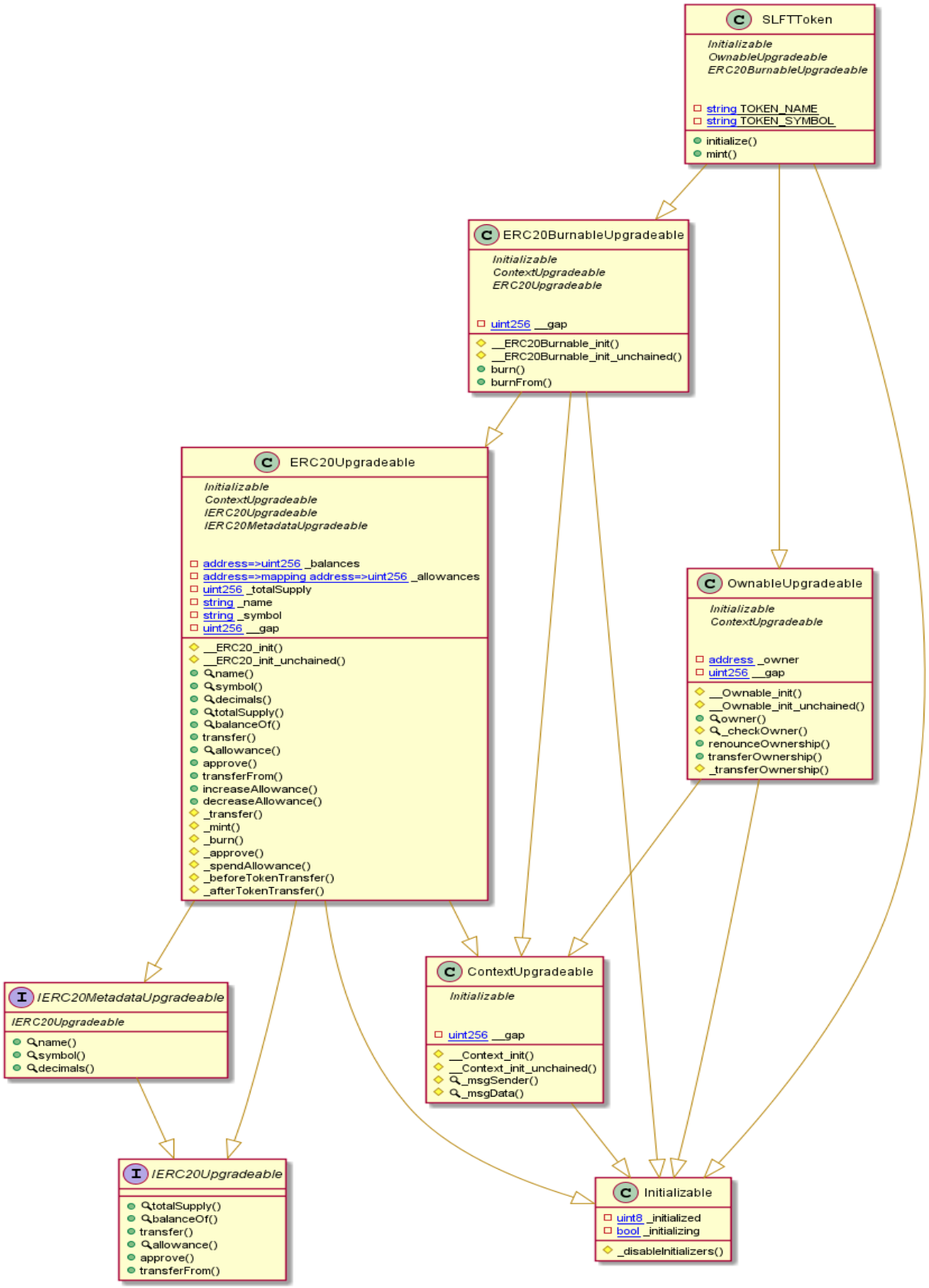
Technical Disclaimer

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 explicit security of the audited smart contracts.

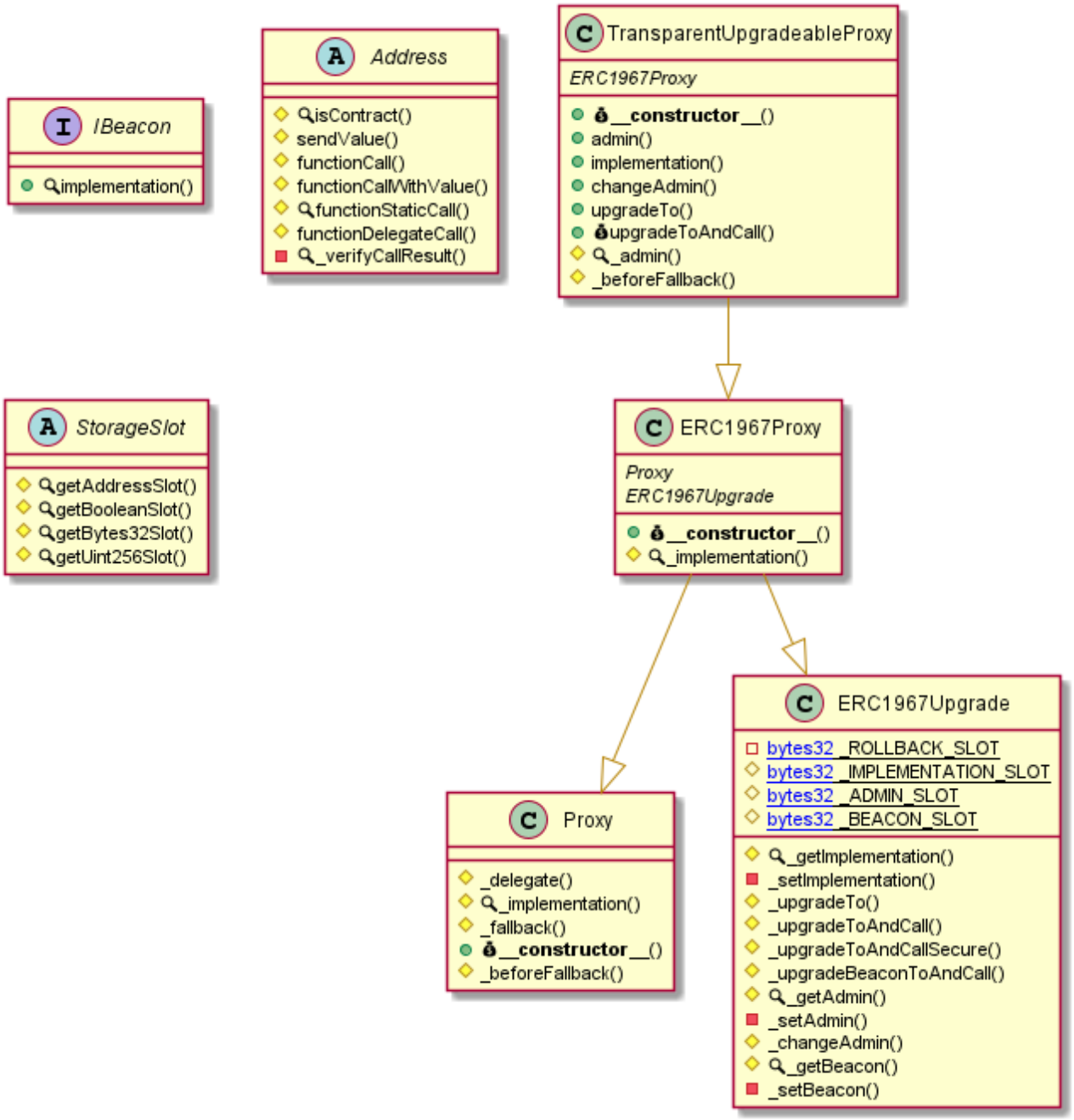
Appendix

Code Flow Diagram - SleeFi Protocol

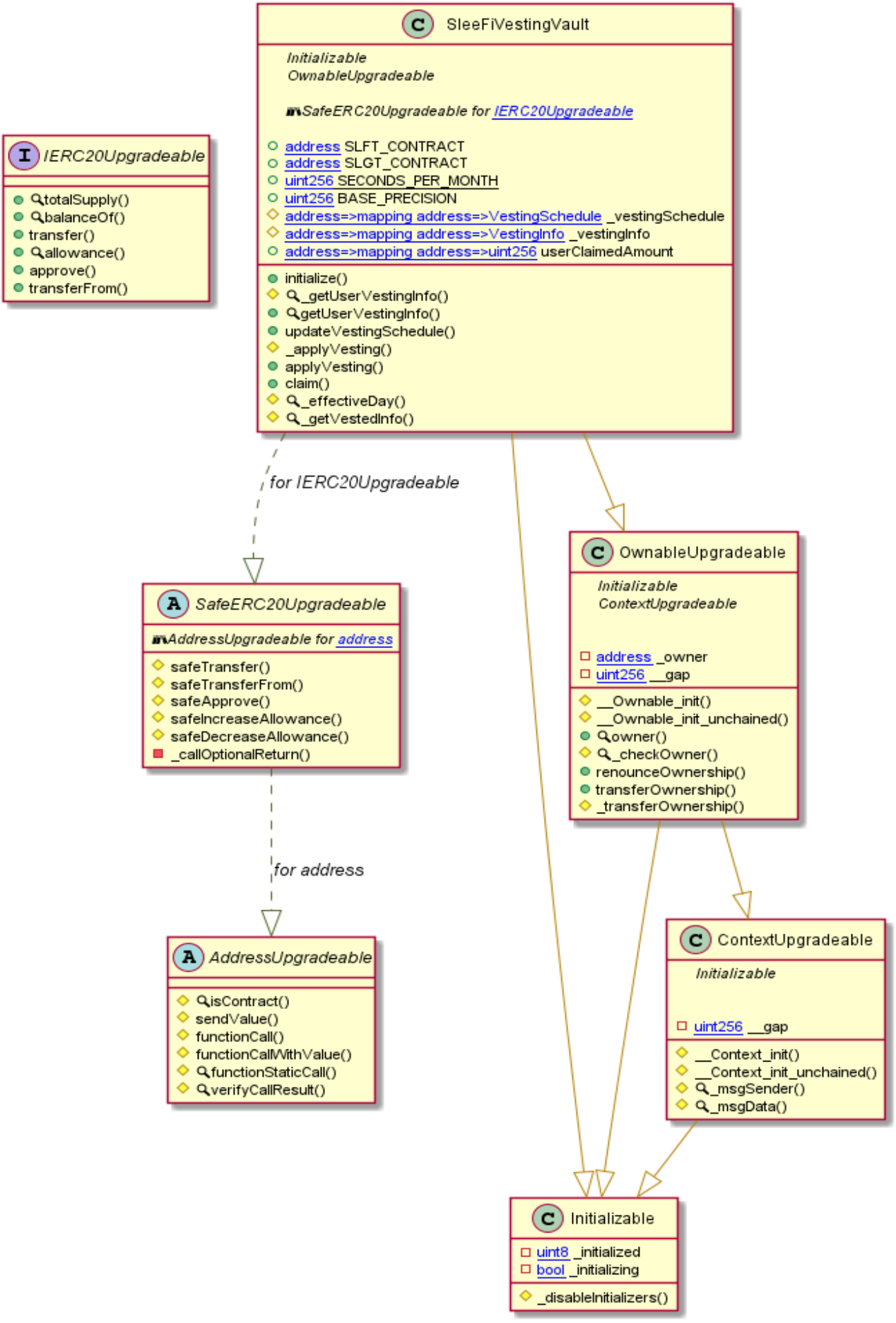
SLFTToken Diagram



SLGTToken Diagram



SleeFiVestingVault Diagram



Slither Results Log

Slither log >> SLFTToken.sol

```
INFO:Detectors:
ContextUpgradeable.__Context_init() (SLFTToken.sol#171-172) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (SLFTToken.sol#174-175) is never used and should be removed
ContextUpgradeable._msgData() (SLFTToken.sol#180-182) is never used and should be removed
ERC20BurnableUpgradeable.__ERC20Burnable_init() (SLFTToken.sol#628-629) is never used and should be removed
ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() (SLFTToken.sol#631-632) is never used and should be removed
Initializable._disableInitializers() (SLFTToken.sol#161-167) 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 (SLFTToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function ContextUpgradeable.__Context_init() (SLFTToken.sol#171-172) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (SLFTToken.sol#174-175) is not in mixedCase
Variable ContextUpgradeable.__gap (SLFTToken.sol#189) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (SLFTToken.sol#199-201) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (SLFTToken.sol#203-205) is not in mixedCase
Variable OwnableUpgradeable.__gap (SLFTToken.sol#264) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (SLFTToken.sol#285-287) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (SLFTToken.sol#289-292) is not in mixedCase
Variable ERC20Upgradeable.__gap (SLFTToken.sol#624) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init() (SLFTToken.sol#628-629) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() (SLFTToken.sol#631-632) is not in mixedCase
Variable ERC20BurnableUpgradeable.__gap (SLFTToken.sol#663) is not in mixedCase
Parameter SLFTToken.mint(address,uint256)._to (SLFTToken.sol#676) is not in mixedCase
Parameter SLFTToken.mint(address,uint256)._amount (SLFTToken.sol#676) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ERC20BurnableUpgradeable.__gap (SLFTToken.sol#663) is never used in SLFTToken (SLFTToken.sol#667-680)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
```

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (SLFTToken.sol#236-238)
transferOwnership(address) should be declared external:
        - OwnableUpgradeable.transferOwnership(address) (SLFTToken.sol#244-247)
name() should be declared external:
        - ERC20Upgradeable.name() (SLFTToken.sol#297-299)
symbol() should be declared external:
        - ERC20Upgradeable.symbol() (SLFTToken.sol#305-307)
decimals() should be declared external:
        - ERC20Upgradeable.decimals() (SLFTToken.sol#322-324)
totalSupply() should be declared external:
        - ERC20Upgradeable.totalSupply() (SLFTToken.sol#329-331)
balanceOf(address) should be declared external:
        - ERC20Upgradeable.balanceOf(address) (SLFTToken.sol#336-338)
transfer(address,uint256) should be declared external:
        - ERC20Upgradeable.transfer(address,uint256) (SLFTToken.sol#348-352)
approve(address,uint256) should be declared external:
        - ERC20Upgradeable.approve(address,uint256) (SLFTToken.sol#371-375)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Upgradeable.transferFrom(address,address,uint256) (SLFTToken.sol#393-402)
increaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.increaseAllowance(address,uint256) (SLFTToken.sol#416-420)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.decreaseAllowance(address,uint256) (SLFTToken.sol#436-445)
burn(uint256) should be declared external:
        - ERC20BurnableUpgradeable.burn(uint256) (SLFTToken.sol#638-640)
burnFrom(address,uint256) should be declared external:
        - ERC20BurnableUpgradeable.burnFrom(address,uint256) (SLFTToken.sol#653-656)
mint(address,uint256) should be declared external:
        - SLFTToken.mint(address,uint256) (SLFTToken.sol#676-678)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:SLFTToken.sol analyzed (8 contracts with 75 detectors), 41 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> SLGTToken.sol

```
INFO:Detectors:
ContextUpgradeable.__Context_init() (SLGTToken.sol#171-172) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (SLGTToken.sol#174-175) is never used and should be removed
ContextUpgradeable._msgData() (SLGTToken.sol#180-182) is never used and should be removed
ERC20Upgradeable._burn(address,uint256) (SLGTToken.sol#515-530) is never used and should be removed
Initializable._disableInitializers() (SLGTToken.sol#161-167) 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 (SLGTToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function ContextUpgradeable.__Context_init() (SLGTToken.sol#171-172) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (SLGTToken.sol#174-175) is not in mixedCase
Variable ContextUpgradeable.__gap (SLGTToken.sol#189) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (SLGTToken.sol#199-201) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (SLGTToken.sol#203-205) is not in mixedCase
Variable OwnableUpgradeable.__gap (SLGTToken.sol#264) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (SLGTToken.sol#285-287) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (SLGTToken.sol#289-292) is not in mixedCase
Variable ERC20Upgradeable.__gap (SLGTToken.sol#624) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ERC20Upgradeable.__gap (SLGTToken.sol#624) is never used in SLGTToken (SLGTToken.sol#628-639)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
```

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (SLGTToken.sol#236-238)
transferOwnership(address) should be declared external:
        - OwnableUpgradeable.transferOwnership(address) (SLGTToken.sol#244-247)
name() should be declared external:
        - ERC20Upgradeable.name() (SLGTToken.sol#297-299)
symbol() should be declared external:
        - ERC20Upgradeable.symbol() (SLGTToken.sol#305-307)
decimals() should be declared external:
        - ERC20Upgradeable.decimals() (SLGTToken.sol#322-324)
totalSupply() should be declared external:
        - ERC20Upgradeable.totalSupply() (SLGTToken.sol#329-331)
balanceOf(address) should be declared external:
        - ERC20Upgradeable.balanceOf(address) (SLGTToken.sol#336-338)
transfer(address,uint256) should be declared external:
        - ERC20Upgradeable.transfer(address,uint256) (SLGTToken.sol#348-352)
approve(address,uint256) should be declared external:
        - ERC20Upgradeable.approve(address,uint256) (SLGTToken.sol#371-375)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Upgradeable.transferFrom(address,address,uint256) (SLGTToken.sol#393-402)
increaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.increaseAllowance(address,uint256) (SLGTToken.sol#416-420)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.decreaseAllowance(address,uint256) (SLGTToken.sol#436-445)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:SLGTToken.sol analyzed (7 contracts with 75 detectors), 31 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> SleeFiVestingVault.sol

```
INFO:Detectors:
SleeFiVestingVault.initialize(address,address)._slft (SleeFiVestingVault.sol#591) lacks a zero-check on :
                - SLFT_CONTRACT = _slft (SleeFiVestingVault.sol#594)
SleeFiVestingVault.initialize(address,address)._slgt (SleeFiVestingVault.sol#591) lacks a zero-check on :
                - SLGT_CONTRACT = _slgt (SleeFiVestingVault.sol#595)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in SleeFiVestingVault.applyVesting(address,address,uint256,uint256,uint256,uint256,uint256) (SleeFiVestingVault.sol#
712-740):
        External calls:
        - _applyVesting(beneficiary,vestingToken,vestingAmount,startTime,cliffDuration,duration,unlockPercent) (SleeFiVestingVa
ult.sol#721-729)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (SleeFiVestingVault.sol#344)
                - (success,returndata) = target.call{value: value}(data) (SleeFiVestingVault.sol#209)
                - IERC20Upgradeable(token).safeTransferFrom(msg.sender,address(this),amount) (SleeFiVestingVault.sol#705-709)
        External calls sending eth:
        - _applyVesting(beneficiary,vestingToken,vestingAmount,startTime,cliffDuration,duration,unlockPercent) (SleeFiVestingVa
ult.sol#721-729)
                - (success,returndata) = target.call{value: value}(data) (SleeFiVestingVault.sol#209)
        Event emitted after the call(s):
        - VestingApplied(beneficiary,vestingToken,vestingAmount,startTime,cliffDuration,duration,unlockPercent) (SleeFiVestingV
ault.sol#731-739)
Reentrancy in SleeFiVestingVault.claim(address) (SleeFiVestingVault.sol#742-767):
        External calls:
        - IERC20Upgradeable(token).transfer(msg.sender,claimAmount) (SleeFiVestingVault.sol#764)
        Event emitted after the call(s):
        - Claimed(msg.sender,token,claimAmount) (SleeFiVestingVault.sol#766)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
SleeFiVestingVault.updateVestingSchedule(address,address,uint256,uint256,uint256,uint256) (SleeFiVestingVault.sol#650-678) uses
 timestamp for comparisons
```

```
        Dangerous comparisons:
        - require(bool,string)(userSchedule.startTime > 0 && block.timestamp < userSchedule.startTime,No vesting info or vestin
g started) (SleeFiVestingVault.sol#661-664)
SleeFiVestingVault._getVestedInfo(uint256,uint256,uint256,uint256,uint256,uint256) (SleeFiVestingVault.sol#778-814) uses timest
amp for comparisons
        Dangerous comparisons:
        - onDay < _startTime + _cliffDuration (SleeFiVestingVault.sol#790)
        - onDay >= _startTime + (_cliffDuration + _duration) (SleeFiVestingVault.sol#795)
        - vestedAmount > _grantedAmount (SleeFiVestingVault.sol#808)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
AddressUpgradeable.verifyCallResult(bool,bytes,string) (SleeFiVestingVault.sol#246-266) uses assembly
        - INLINE ASM (SleeFiVestingVault.sol#258-261)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
AddressUpgradeable.functionCall(address,bytes) (SleeFiVestingVault.sol#157-159) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (SleeFiVestingVault.sol#186-192) is never used and should be re
moved
AddressUpgradeable.functionStaticCall(address,bytes) (SleeFiVestingVault.sol#219-221) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (SleeFiVestingVault.sol#229-238) is never used and should be remove
d
AddressUpgradeable.sendValue(address,uint256) (SleeFiVestingVault.sol#132-137) is never used and should be removed
ContextUpgradeable.__Context_init() (SleeFiVestingVault.sol#432-433) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (SleeFiVestingVault.sol#435-436) is never used and should be removed
ContextUpgradeable._msgData() (SleeFiVestingVault.sol#441-443) is never used and should be removed
Initializable._disableInitializers() (SleeFiVestingVault.sol#422-428) is never used and should be removed
SafeERC20Upgradeable.safeApprove(IERC20Upgradeable,address,uint256) (SleeFiVestingVault.sol#296-309) is never used and should b
e removed
SafeERC20Upgradeable.safeDecreaseAllowance(IERC20Upgradeable,address,uint256) (SleeFiVestingVault.sol#320-331) is never used an
d should be removed
SafeERC20Upgradeable.safeIncreaseAllowance(IERC20Upgradeable,address,uint256) (SleeFiVestingVault.sol#311-318) is never used an
d should be removed
SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable,address,uint256) (SleeFiVestingVault.sol#272-278) 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 (SleeFiVestingVault.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
```

```
INFO:Detectors:
Pragma version^0.8.0 (SleeFiVestingVault.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in AddressUpgradeable.sendValue(address,uint256) (SleeFiVestingVault.sol#132-137):
        - (success) = recipient.call{value: amount}() (SleeFiVestingVault.sol#135)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (SleeFiVestingVault.sol#200-211):
        - (success,returndata) = target.call{value: value}(data) (SleeFiVestingVault.sol#209)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (SleeFiVestingVault.sol#229-238):
        - (success,returndata) = target.staticcall(data) (SleeFiVestingVault.sol#236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function ContextUpgradeable.__Context_init() (SleeFiVestingVault.sol#432-433) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (SleeFiVestingVault.sol#435-436) is not in mixedCase
Variable ContextUpgradeable.__gap (SleeFiVestingVault.sol#450) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (SleeFiVestingVault.sol#460-462) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (SleeFiVestingVault.sol#464-466) is not in mixedCase
Variable OwnableUpgradeable.__gap (SleeFiVestingVault.sol#525) is not in mixedCase
Parameter SleeFiVestingVault.initialize(address,address)._slft (SleeFiVestingVault.sol#591) is not in mixedCase
Parameter SleeFiVestingVault.initialize(address,address)._slgt (SleeFiVestingVault.sol#591) is not in mixedCase
Variable SleeFiVestingVault.SLFT_CONTRACT (SleeFiVestingVault.sol#558) is not in mixedCase
Variable SleeFiVestingVault.SLGT_CONTRACT (SleeFiVestingVault.sol#559) is not in mixedCase
Variable SleeFiVestingVault.BASE_PRECISION (SleeFiVestingVault.sol#562) is not in mixedCase
Variable SleeFiVestingVault._vestingSchedule (SleeFiVestingVault.sol#564) is not in mixedCase
Variable SleeFiVestingVault._vestingInfo (SleeFiVestingVault.sol#565) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable SleeFiVestingVault.SLFT_CONTRACT (SleeFiVestingVault.sol#558) is too similar to SleeFiVestingVault.SLGT_CONTRACT (Slee
FiVestingVault.sol#559)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
OwnableUpgradeable.__gap (SleeFiVestingVault.sol#525) is never used in SleeFiVestingVault (SleeFiVestingVault.sol#529-816)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
```

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (SleeFiVestingVault.sol#497-499)
transferOwnership(address) should be declared external:
        - OwnableUpgradeable.transferOwnership(address) (SleeFiVestingVault.sol#505-508)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:SleeFiVestingVault.sol analyzed (7 contracts with 75 detectors), 45 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

SLFTToken.sol

Gas & Economy

Gas costs:

Gas requirement of function SLFTToken.initialize 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: 671:4:

Gas costs:

Gas requirement of function SLFTToken.mint 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: 676:4:

Miscellaneous

Constant/View/Pure functions:

ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 631:4:

Similar variable names:

ERC20BurnableUpgradeable.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 655:23:

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: 572:12:

SLGTToken.sol

Gas & Economy

Gas costs:

Gas requirement of function SLGTToken.initialize 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: 632:4:

Miscellaneous

Constant/View/Pure functions:

ERC20Upgradeable._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 613:4:

Similar variable names:

ERC20Upgradeable._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 529:49:

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: 572:12:

SleeFiVestingVault.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SleeFiVestingVault.claim(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 742:4:

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: 775:35:

Gas & Economy

Gas costs:

Gas requirement of function SleeFiVestingVault.initialize 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: 591:4:

Miscellaneous

Constant/View/Pure functions:

SleeFiVestingVault._applyVesting(address,address,uint256,uint256,uint256,uint256,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 680:4:

Similar variable names:

SleeFiVestingVault.claim(address) : Variables have very similar names "claimedAmount" and "claimAmount". Note: Modifiers are currently not considered by this static analysis.
Pos: 766:40:

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: 751:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 803:35:

Solhint Linter

SLFTToken.sol

```
SLFTToken.sol:440:18: Error: Parse error: missing ';' at '{'
SLFTToken.sol:473:18: Error: Parse error: missing ';' at '{'
SLFTToken.sol:522:18: Error: Parse error: missing ';' at '{'
SLFTToken.sol:573:22: Error: Parse error: missing ';' at '{'
```

SLGTToken.sol

```
SLGTToken.sol:440:18: Error: Parse error: missing ';' at '{'
SLGTToken.sol:473:18: Error: Parse error: missing ';' at '{'
SLGTToken.sol:522:18: Error: Parse error: missing ';' at '{'
SLGTToken.sol:573:22: Error: Parse error: missing ';' at '{'
```

SleeFiVestingVault.sol

```
SleeFiVestingVault.sol:325:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

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

Email: audit@EtherAuthority.io