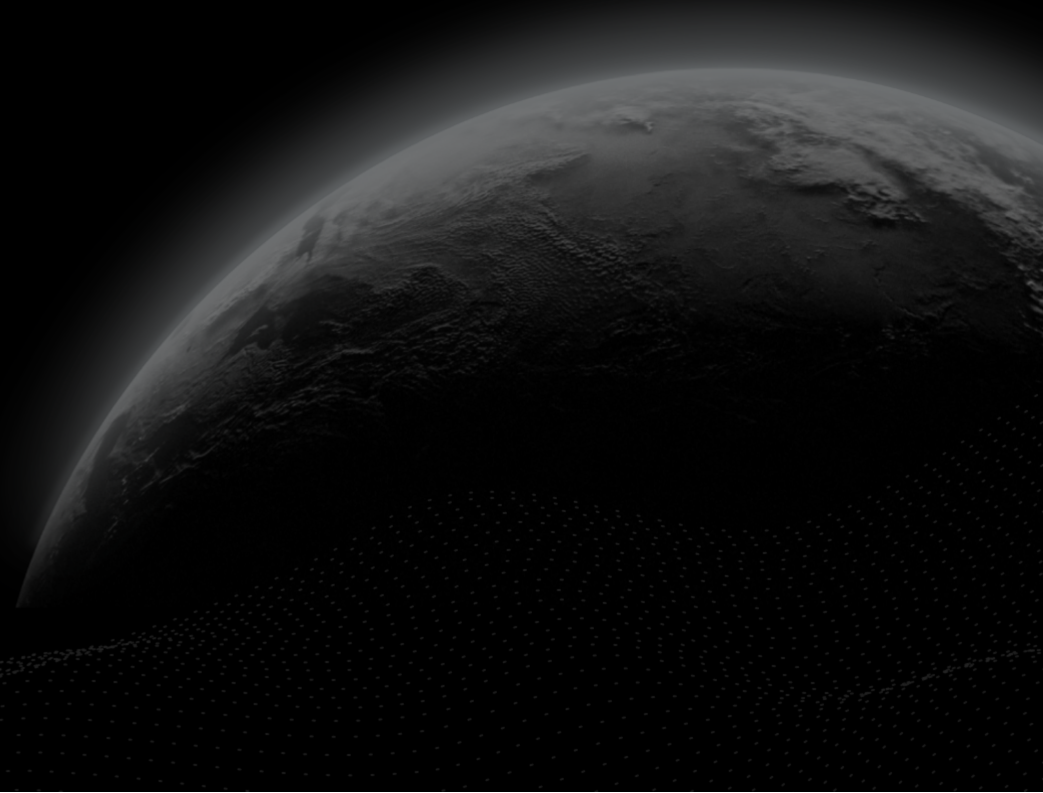
CERTIK

Security Assessment

FILLiquid

CertiK Assessed on Apr 25th, 2024



CertiK Assessed on Apr 25th, 2024

FILLiquid

The security assessment was prepared by CertiK, the leader in Web3.0 security.

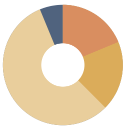
Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Filecoin (FIL)	Formal Verification, Manual Review, Static Analysis
LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 04/25/2024	N/A

CODEBASE
https://github.com/FILL-Lab/FILLiquid/
View All in Codebase Page

COMMITS
87d212ecce911e0e44a8df00bd82c3917cc5e261
View All in Codebase Page

Vulnerability Summary



16	11	1	4	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧 3	Major	2 Resolved, 1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 3	Medium	3 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 9	Minor	6 Resolved, 3 Partially Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 1	Informational	1 Partially Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | FILLIQUID

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Findings

FIL-02 : Initial Token Distribution

FIL-03 : Centralized Balance Manipulation

FIL-04 : Centralization Related Risks

FIF-03 : Check Effect Interaction Pattern - FIL transfer

MSF-02 : Missing Check of Duplicate Signer Address

MSF-03 : Exclusion of `msg.value` in Proposal and Unchecked Low Level Call Could Cause Failure of Proposal Execution

CUF-01 : Inconsistency with Comments

FIF-02 : Potential Frontrunning Attack if `expectAmountFILTrust` and `expectAmountFIL` Are Not Set Properly in the Deposit and Redeem Functions

FIL-05 : Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call

FIL-06 : Missing Zero Address Validation

FIL-07 : Check Effect Interaction Pattern Violated (Out-of-Order Events)

FIS-02 : Missing check on `FILStake.setStakeParams()`

FIS-03 : `FILStake.setShares()` allows zero values

FIS-04 : Checks Effects Interaction Pattern Not Used

MSF-04 : Missing Threshold Requirement in Multisig

FIL-08 : Missing Emit Events

Optimizations

FIL-01 : Variables That Could Be Declared as Immutable

FIS-01 : If Statement Optimization

MSF-01 : User-Defined Getters

Formal Verification

Considered Functions And Scope

Verification Results

Appendix

Disclaimer

CODEBASE | FILLIQUID

Repository

https://github.com/FILL-Lab/FILLiquid/

Commit

87d212ecce911e0e44a8df00bd82c3917cc5e261

AUDIT SCOPE | FILLIQUID

13 files audited ● 6 files with Mitigated findings ● 1 file with Partially Resolved findings ● 1 file with Resolved findings ● 5 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● ERC	FILL-Lab/FILLiquid	87d212e	contracts/ERC20Pot.sol	8d5a8d6afdb38d53a8f0746eeec423a24a3dc898de23388a12ddc7964294fdf9
● FIL	FILL-Lab/FILLiquid	87d212e	contracts/FILGovernance.sol	4beaff266fb7b249a8f51e6979151b4d21d71c433ac905bcc91c894034ac65ff
● FIF	FILL-Lab/FILLiquid	87d212e	contracts/FILLiquid.sol	afc78a0ef3f86cecdc0be7ffd8981deeeeef358afd5364250a0432ea55815279
● FIS	FILL-Lab/FILLiquid	87d212e	contracts/FILStake.sol	bd511a0a179e619f17a5453e7bce740151ef846c016ef600fa4a5232d442a9d6
● FIT	FILL-Lab/FILLiquid	87d212e	contracts/FILTrust.sol	7f55f3bc9f4be0e2bea01726d4de9398c55b03b07c07c30e6dc7eae5d023ad14
● GFI	FILL-Lab/FILLiquid	87d212e	contracts/Governance.sol	1be7e939173271c889df509018f1de5f63af5d7230a576c27fcdf877e1a5a1be
● MSF	FILL-Lab/FILLiquid	87d212e	contracts/MultiSignFactory.sol	463c8fd3648cdd9527046c3966c253a4c4fe0f43ab6b3f3aecad7594c990ad4a
● CUF	FILL-Lab/FILLiquid	87d212e	contracts/Utils/Calculation.sol	a20619d0a3f46bb42892ccb5c0257ef238359fce9ff8dca3805b7f35d9eff997
● CUI	FILL-Lab/FILLiquid	87d212e	contracts/Utils/Conversion.sol	1413a278950ded2aa2d9cb3b8352897e3a800fd5c0e2510ad32e26177ec906c7
● FAP	FILL-Lab/FILLiquid	87d212e	contracts/Utils/FilecoinAPI.sol	2eb0d34e22efab2d7c53952d0544e835b4cf5a9ca675db1d93b6090080bf11d3
● VUF	FILL-Lab/FILLiquid	87d212e	contracts/Utils/Validation.sol	3df8107a2c808e78fdc9537a56c04e1a0e2bb7426b6c0fa2bf6e279bf984aafa
● DFI	FILL-Lab/FILLiquid	b6ce507	contracts/Deployer1.sol	8aa6c22435b1243f97175855c02cc3bd2305066c5aa4ab16b3f48f1f563f0ead

ID	Repo	Commit	File	SHA256 Checksum
DFL	FILL-Lab/FILLiquid	b6ce507	contracts/Deployer2.sol	4947dc401d2f7e0209e954e5212850a29de44387b5f72fcd2f53b508e75a7230

APPROACH & METHODS | FILLIQUID

This report has been prepared for FILLiquid to discover issues and vulnerabilities in the source code of the FILLiquid project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

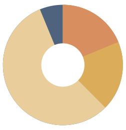
The auditing process pays special attention to the following considerations:

- 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.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | FILLIQUID



16	0	3	3	9	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for FILLiquid. Through this audit, we have uncovered 16 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
FIL-02	**Initial Token Distribution**	**Centralization**	**Major**	**Resolved**
FIL-03	**Centralized Balance Manipulation**	**Centralization**	**Major**	**Resolved**
FIL-04	**Centralization Related Risks**	**Centralization**	**Major**	**Mitigated**
FIF-03	Check Effect Interaction Pattern - FIL Transfer	Concurrency	Medium	Resolved
MSF-02	Missing Check Of Duplicate Signer Address	Volatile Code	Medium	Resolved
MSF-03	Exclusion Of `msg.value` In Proposal And Unchecked Low Level Call Could Cause Failure Of Proposal Execution	Volatile Code, Logical Issue	Medium	Resolved
CUF-01	Inconsistency With Comments	Inconsistency	Minor	Resolved
FIF-02	Potential Frontrunning Attack If `expectAmountFILTrust` And `expectAmountFIL` Are Not Set Properly In The Deposit And Redeem Functions	Volatile Code, Financial Manipulation	Minor	Partially Resolved
FIL-05	Unchecked Value Of ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
FIL-06	Missing Zero Address Validation	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
FIL-07	Check Effect Interaction Pattern Violated (Out-Of-Order Events)	Concurrency	Minor	● Partially Resolved
FIS-02	Missing Check On `FILStake.setStakeParams()`	Logical Issue	Minor	● Resolved
FIS-03	`FILStake.setShares()` Allows Zero Values	Logical Issue	Minor	● Resolved
FIS-04	Checks Effects Interaction Pattern Not Used	Volatile Code	Minor	● Partially Resolved
MSF-04	Missing Threshold Requirement In Multisig	Volatile Code	Minor	● Resolved
FIL-08	Missing Emit Events	Coding Style	Informational	● Partially Resolved

FIL-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● Major	contracts/FILGovernance.sol (base): 16~17	● Resolved

Description

40% of the `FILGovernance` tokens are sent to the contract deployer/owner. This is a centralization risk because the deployer / owner can significantly impact the governance system implemented in the `Governance` contract. Additionally, the deployer / owner can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project and indirectly impacts the Governance system.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[CertiK, 2024/04/11]: Based on the deployment contract `Deploy1.sol` at commit b6ce507a20a2f517b66f3a3785aa0bc5dd543ef4, 40% of the `FILGovernance` tokens are transferred to the multisig `MultiSignFactory.sol` when the token is deployed.

[CertiK, 2024/04/25]: Based on the deployment contract `Deploy1.sol` at commit c0ea7b5ca905875d1d9c4df1115827992795c69b, the `FILGovernance` tokens are distributed according to the following constant percentages and are subject to a fixed vesting schedule as defined in the `ERC20Pot` contract. Given the immutable nature of the distribution % and vesting schedule, we consider the issue "resolved".

```
    uint constant INSTITUTION_LOCKING_PERIOD = 1036800; //360 days
    uint constant TEAM_LOCKING_PERIOD = 3110400; //1080 days
    uint constant FOUNDATION_LOCKING_PERIOD = 3110400; //1080 days
    uint constant RESERVE_LOCKING_PERIOD = 1036800; //360 days
    uint constant COMMUNITY_LOCKING_PERIOD = 259200; //90 days

    uint constant INSTITUTION_SHARE = 250;
    uint constant TEAM_SHARE = 375;
    uint constant FOUNDATION_SHARE = 125;
    uint constant RESERVE_SHARE = 125;
    uint constant COMMUNITY_SHARE = 125;
    uint constant RATEBASE = 1000;
```

FIL-03 | CENTRALIZED BALANCE MANIPULATION

Category	Severity	Location	Status
Centralization	● **Major**	**contracts/FILGovernance.sol (base): 20~26; contracts/FILTrust.sol (base): 14~20**	● **Resolved**

Description

In the contract `FILGovernance` and `FILTrust`, the role `Owner` has the authority to add/remove `Manager`, and the `Manager` can withdraw token balance of an arbitrary account without restriction.

Any compromise to the `Owner` or Manager` account may allow a hacker to take advantage of this authority and manipulate users' balances. The hacker can subsequently use the tokens to directly impact the Governance system and potentially execute malicious proposals, or sell the tokens on the market.

Recommendation

Given that the `withdraw()` function is only intended to be called by the `FILStake` and `Governance` contracts, consider using an immutable check that the caller of the function is either the `FILStake` or `Governance` contract instead.

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to mint more tokens or engage in similar balance-related operations.
Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently *fully* resolve the risk:

Short Term:

A multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers' addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR
- Remove the risky functionality.
  OR
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[CertiK, 2024/04/11]: Based on the deployment contract `Deploy1.sol` and `Deploy2.sol` at commit b6ce507a20a2f517b66f3a3785aa0bc5dd543ef4, the `setting()` function of `Deploy1.sol` would transfer the ownership of the `FILGovernance` and `FILTrust` tokens to the address `deployer2`. If the `deployer2` address is the `Deploy2.sol` contract, AND both deployment contracts are configured correctly and the `setting()` function is called on the `Deploy2.sol` contract, then the only `Manager` of the `FILTrust` token would be `_filLiquid` and `_fitStake`, and the only `Manager` of the `FILGovernance` token would be `_fitStake` and `_governance`. Moreover, ownership would be renounced for `_filTrust`, `_filGovernance`, `_fitStake`, `_governance`, and `_filLiquid`.

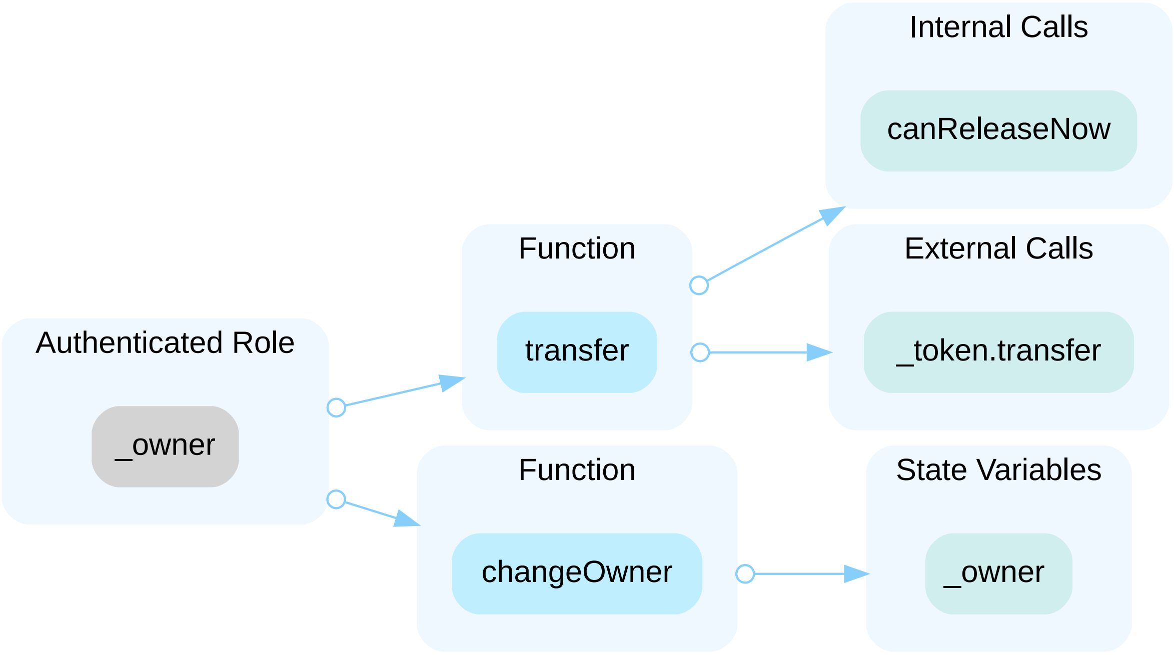
[CertiK, 2024/04/25]: Based on https://github.com/FILL-Lab/FILLiquid/pull/92, despite having the `manager` role of the `FILGovernance` token, the `Governance` contract cannot call the `withdraw()` or `mint()` functions of the `FILGovernance` token, as the `Governance` contract does not allow arbitrary call even after a proposal passes the voting process successfully. As such, the finding can be considered "resolved".

FIL-04 | CENTRALIZATION RELATED RISKS

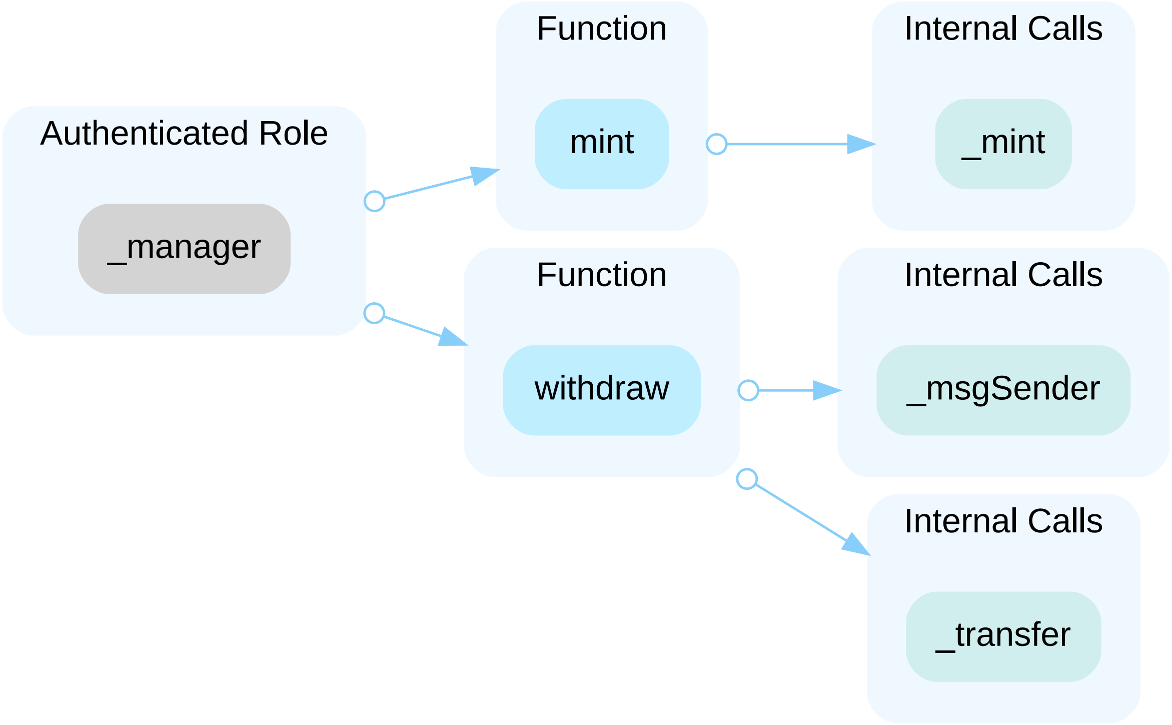
Category	Severity	Location	Status
Centralization	● Major	contracts/ERC20Pot.sol (base): 30, 36; contracts/FILGovernance.sol (base): 20, 24, 28, 40, 44, 56, 65; contracts/FILLiquid.sol (base): 805, 821, 923; contracts/FILStake.sol (base): 242, 249, 256, 273, 291; contracts/FILTrust.sol (base): 14, 18, 22, 27, 31, 43; contracts/Governance.sol (base): 356, 394, 408	● Mitigated

Description

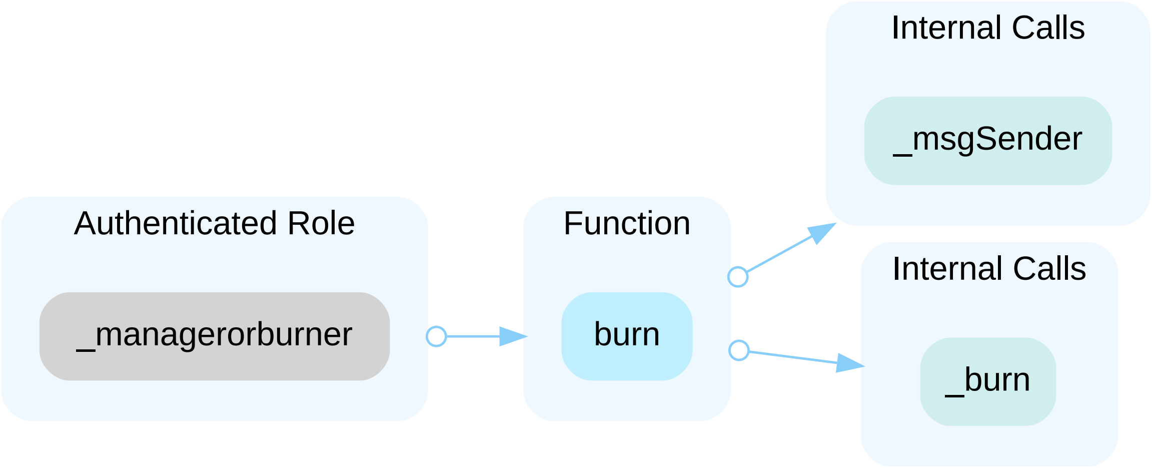
In the contract `ERC20Pot` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and transfer tokens to arbitrary addresses.



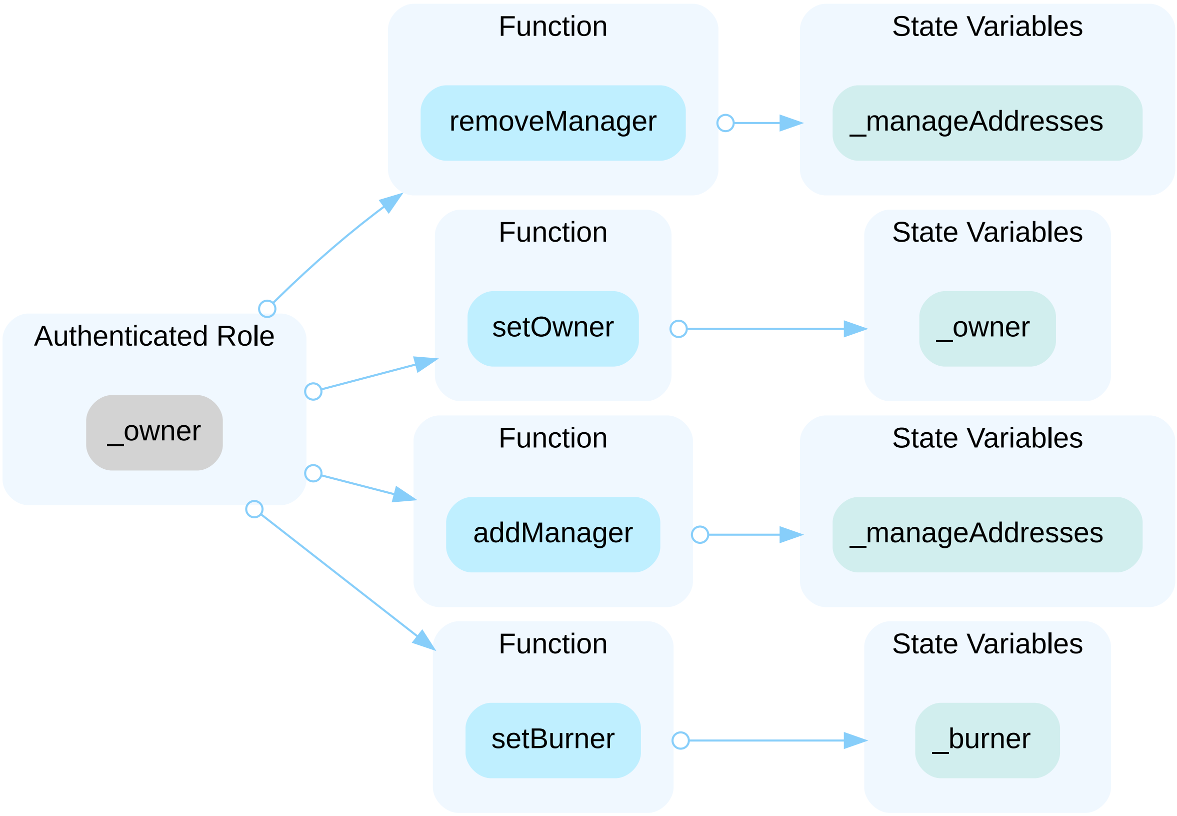
In the contract `FILGovernance` the role `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_manager` account may allow the hacker to take advantage of this authority and transfer tokens from any arbitrary address.



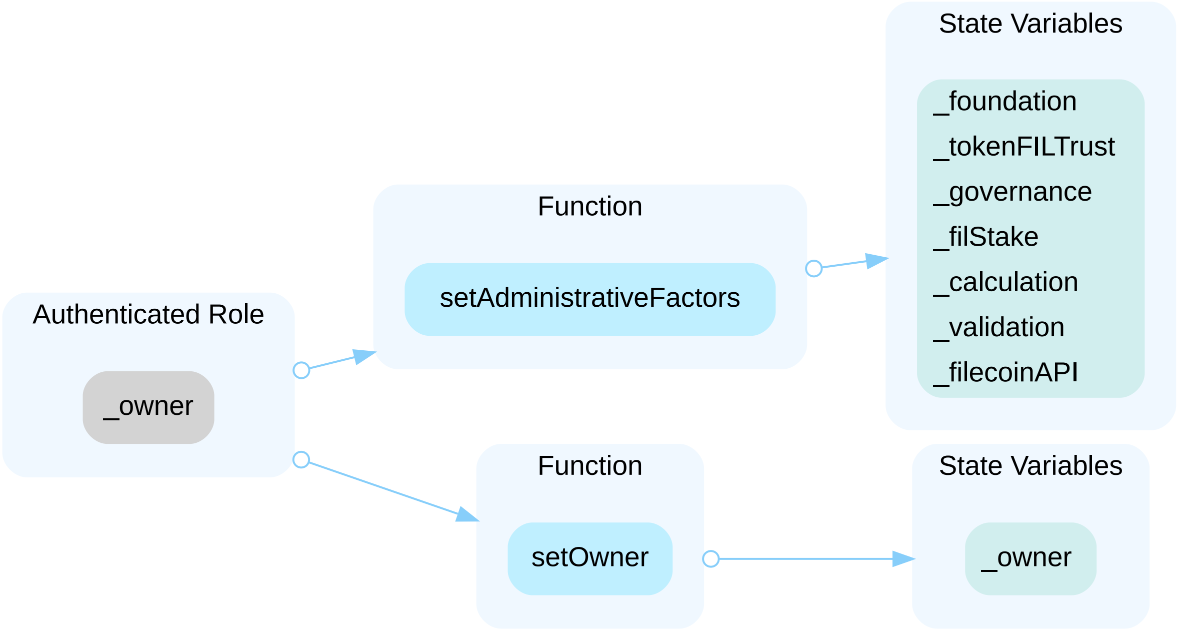
In the contract `FILGovernance` the role `_managerorburner` has authority over the functions shown in the diagram below. Any compromise to the `_managerorburner` account may allow the hacker to take advantage of this authority and burn tokens.



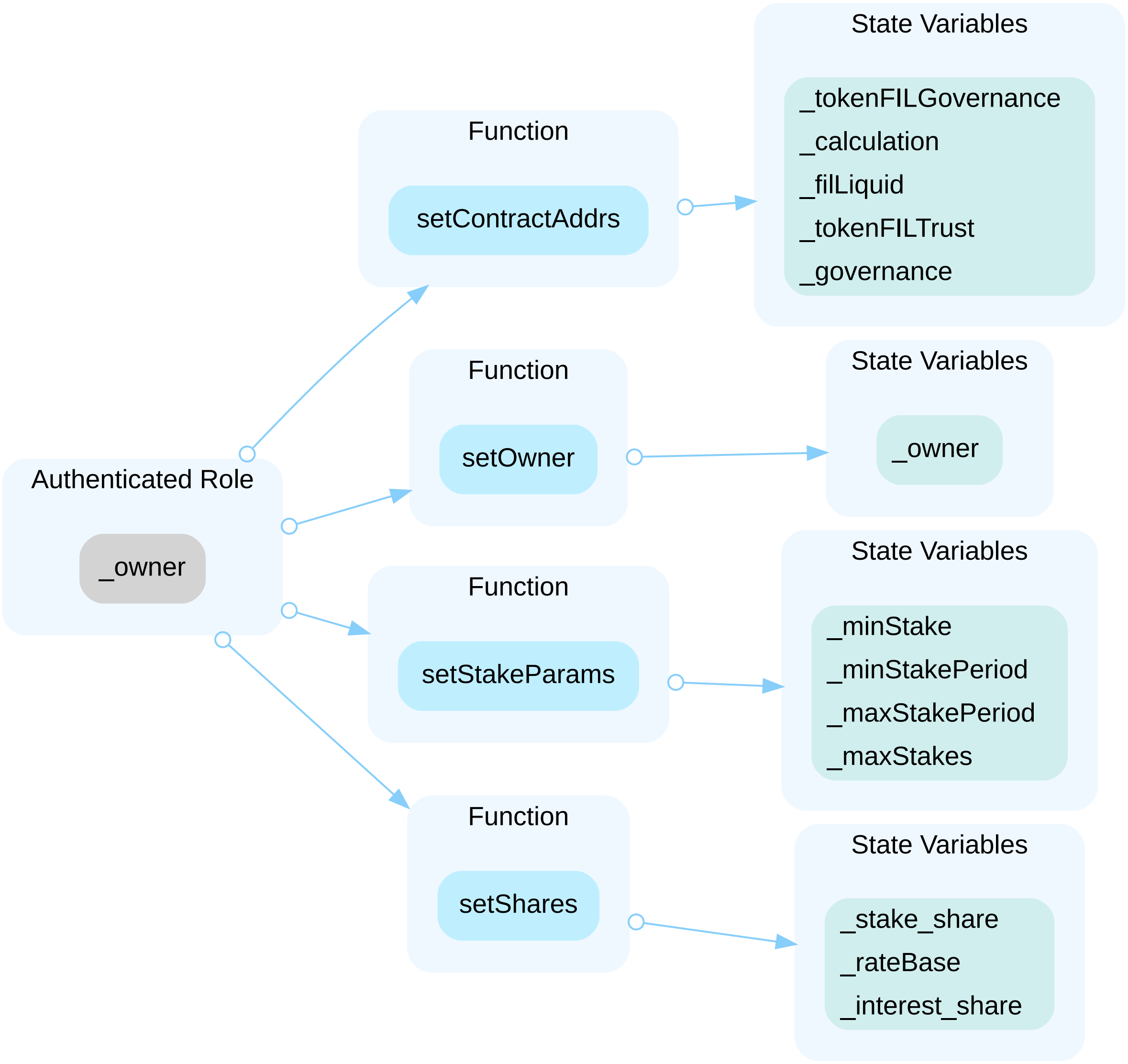
In the contract `FILGovernance` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and add/remove `_manager` and `burner`.



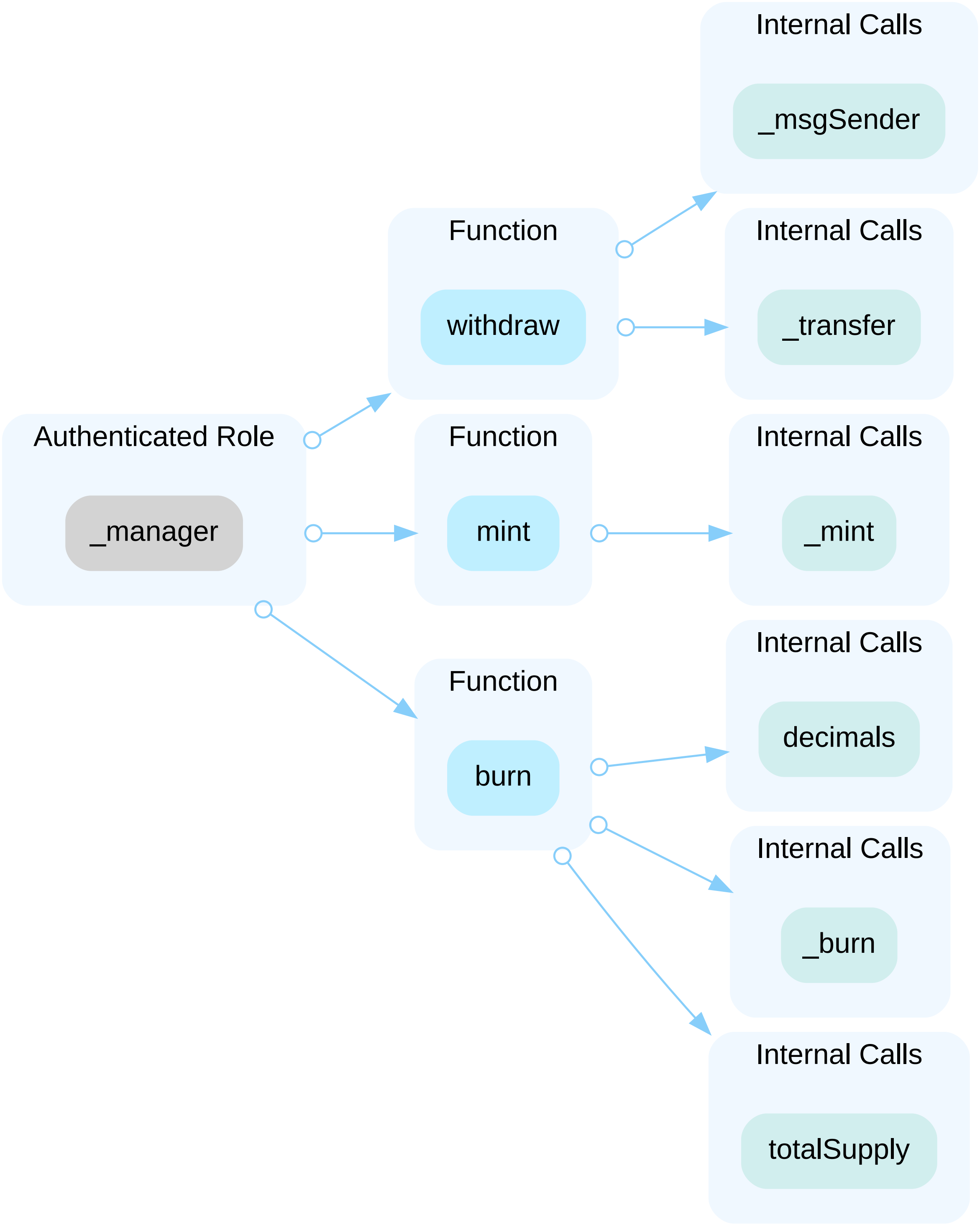
In the contract `FILLiquid` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set important addresses including the `_governance` address which can call the `setGovernanceFactors()` function to change critical parameters.



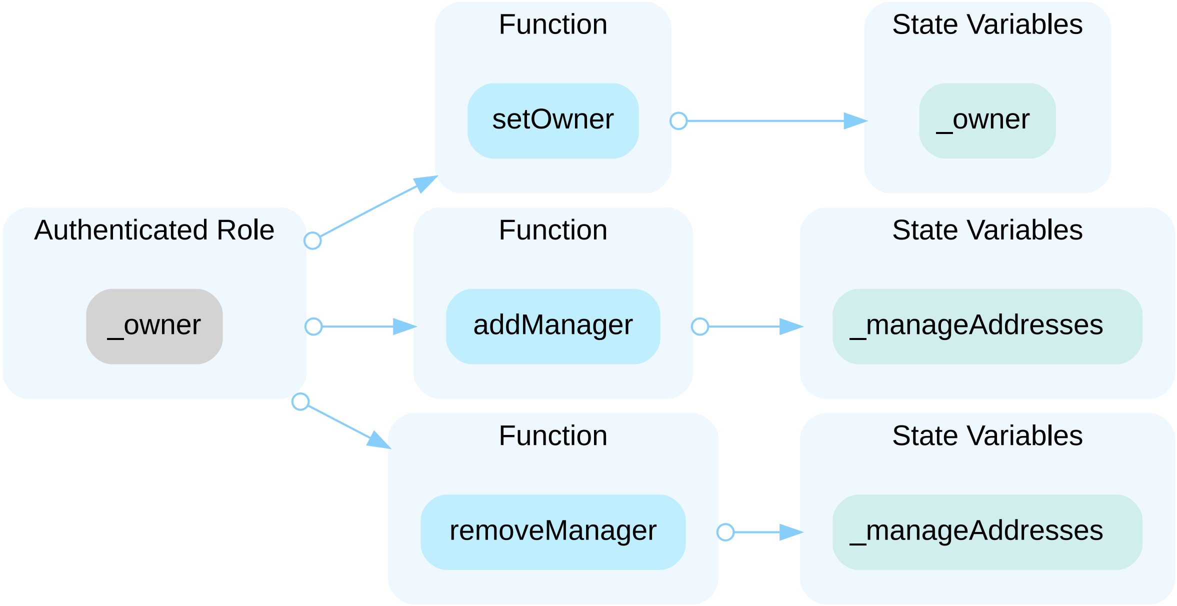
In the contract `FILStake` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set critical addresses and parameters.



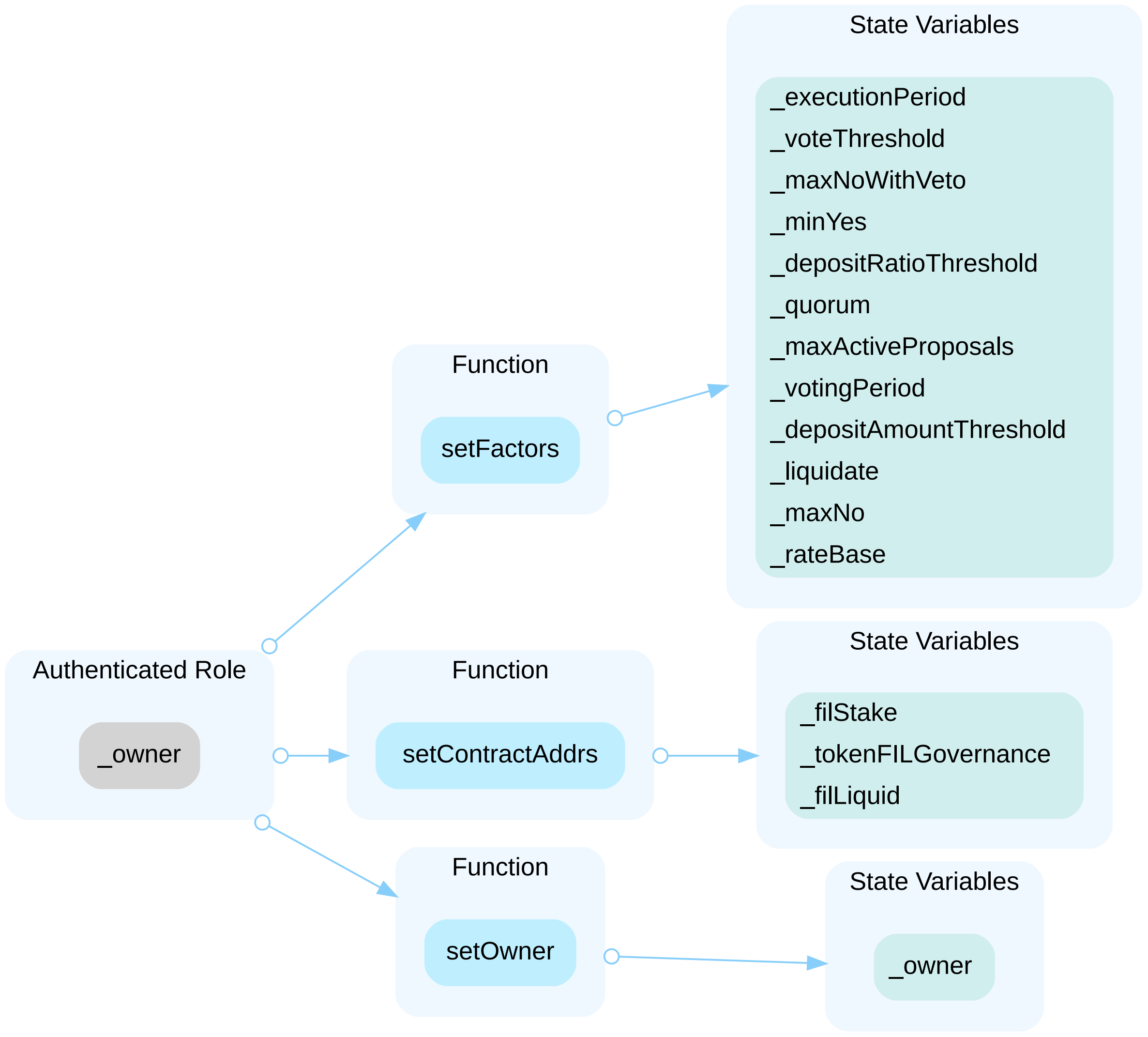
In the contract `FILTrust` the role `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_manager` account may allow the hacker to take advantage of this authority and transfer tokens from arbitrary addresses and mint/burn tokens.



In the contract `FILTrust` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and add/remove `_manager` .



In the contract `Governance` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change important addresses such as the token address that can participate in the governance system, and critical parameters of the Governance system.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key 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, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;

AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[CertiK, 2024/04/11]: Based on the deployment contract `Deploy1.sol` and `Deploy2.sol` at commit b6ce507a20a2f517b66f3a3785aa0bc5dd543ef4, the `setting()` function of `Deploy1.sol` would transfer the ownership of the `FILGovernance` and `FILTrust` tokens to the address `deployer2`. If the `deployer2` address is the `Deploy2.sol` contract, AND both deployment contracts are configured correctly and the `setting()` function is called on the `Deploy2.sol` contract, then the only `Manager` of the `FILTrust` token would be `_filLiquid` and `_fitStake`, and the only `Manager` of the `FILGovernance` token would be `_fitStake` and `_governance`. Moreover, ownership would be renounced for `_filTrust`, `_filGovernance`, `_fitStake`, `_governance`, and `_filLiquid`.

[CertiK, 2024/04/25]: Based on https://github.com/FILL-Lab/FILLiquid/pull/92, if a proposal passes through the governance process, the `Governance` contract can call the `setGovernanceFactors()` function of the `FILLiquid` and `FITStake` contracts, and set important parameters such as liquidation threshold. The `governance` contract is a strong long-term mitigation to the centralization risk.

FIF-03 | CHECK EFFECT INTERACTION PATTERN - FIL TRANSFER

Category	Severity	Location	Status
Concurrency	● Medium	contracts/FILLiquid.sol (base): 402~403, 403~404, 403~404, 450~451, 473~474, 487~488, 513~514, 515~516	● Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects.

If the attacker can control the untrusted contract, they can make a recursive call back to the original contract, repeating interactions that would have otherwise not run after the external call resolved the effects.

In `FILLiquid.sol` , `FIL` tokens are sent to a caller in multiple functions before state variables are updated and events emitted.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Here to follow the Checks-Effects-Interactions pattern, the external calls can be put at the end of the relevant functions, right after updating state variables and events emissions.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

MSF-02 | MISSING CHECK OF DUPLICATE SIGNER ADDRESS

Category	Severity	Location	Status
Volatile Code	● Medium	contracts/MultiSignFactory.sol (base): 61~67, 90~100	● Resolved

Description

A properly set Multisig contract should have unique individual signer addresses. However, there's no such check when the `_signers` are set in the constructor and the `renewSigners()` function of the `MultiSignFactory` contract. If there are duplicate `signer` address in the `signers` array, then the `_approvalThreshold` would not work as intended, as the same signer cannot approve a proposal more than once.

Recommendation

Recommend adding a check that there cannot be duplicate address in the provided `signers` array.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

MSF-03 EXCLUSION OF `msg.value` IN PROPOSAL AND UNCHECKED LOW LEVEL CALL COULD CAUSE FAILURE OF PROPOSAL EXECUTION

Category	Severity	Location	Status
Volatile Code, Logical Issue	● Medium	contracts/MultiSignFactory.sol (base): 69~71, 84~88, 182~201, 234~239	● Resolved

Description

In the `MultiSignFactory` contract, any signer can create a proposal, and if enough signers approve the proposal, any signer can call the `execute()` function to execute the proposal.

When a proposal is created, its `proposer` , `target` , `code` , and `text` are recorded in a `ProposalInfo` struct. When a proposal is executed, the caller can include an arbitrary amount of `msg.value` in its function call. Note that this `msg.value` is not included in the `ProposalInfo` struct. Furthermore, in the `_execute()` function, the return value `success` of the low level call is not checked in line 237, but `info.executed` is always set to `true` regardless. This makes it possible for a proposal that has enough approval to be executed to fail execution due to the `msg.value` being included. For example, calling a nonpayable function of the `target` contract with a positive `msg.value` would fail to execute. The low level call in line 237 would return `false` instead of reverting, and this proposal cannot be re-executed because `info.executed` is already set to `true` in line 236.

Recommendation

Consider including the `msg.value` in the `ProposalInfo` struct when a proposal is created. Also, recommend checking for the return value of the low level call in line 237, and only set `info.executed` to `true` if the `success` variable is `true` .

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

CUF-01 | INCONSISTENCY WITH COMMENTS

Category	Severity	Location	Status
Inconsistency	● Minor	contracts/Utils/Calculation.sol (base): 59~60, 69	● Resolved

Description

In the function `Calculation.getFilByRedeem()` , the `Proportional Redemption` will take place if

```
69          if (filLiquidity * u_m > utilizedLiquidity * rateBase) {
```

however, the comments state that:

```
59
//   - Proportional Redemption when utilizationRate is less than or equal to u_m /
rateBase
```

Recommendation

We recommend modifying the code or the comments accordingly to ensure consistency.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

FIF-02 POTENTIAL FRONTRUNNING ATTACK IF `expectAmountFILTrust` AND `expectAmountFIL` ARE NOT SET PROPERLY IN THE DEPOSIT AND REDEEM FUNCTIONS

Category	Severity	Location	Status
Volatile Code, Financial Manipulation	● Minor	contracts/FILLiquid.sol (base): 375~389, 391~407	● Partially Resolved

Description

In the `FILLiquid` contract, the `deposit()` function takes in a parameter `expectAmountFILTrust` , and the `redeem()` function takes in a parameter `expectAmountFIL` . They act as safety checks for the amount of FIT token and FIL token that the user receives. If they are left at 0 or not set properly, an attacker could potentially frontrun normal user transactions.

For example, when a normal user calls the `redeem()` function when `U < _u_m` , an attacker could frontrun the transaction by borrowing FIL up to `_u_m` , such that when the user redeems, its `U` would be greater than `_u_m` , in which case the curve penalizes redemption and the user would receive a smaller amount of FIL than it otherwise would receive. The attacker could subsequently call `deposit()` and receive a larger share of FIT than normal because of the higher utilization rate, and then payback the borrowing incurring minimal/no interest expense.

Recommendation

We'd like to understand if users are supposed to come up with their own `expectAmountFILTrust` and `expectAmountFIL` , or if the project team would provide those values via off-chain calculation when users interact with the project via the UI. We'd like to understand how the project team calculates those values to protect against potential frontrunning attacks.

Alleviation

Filliquid team:: the calculation and slippage are set off chain, and the slippage parameter would protect against this type of MEV based attacks.

FIL-05 | UNCHECKED VALUE OF ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/FILStake.sol (base): 158; contracts/Governance.sol (base): 164, 229, 241, 244~245	● Resolved

Description

The linked `transfer()` / `transferFrom()` invocations do not check the return value of the function call, which should yield `true` in the case of a proper ERC-20 implementation, including the inherited OpenZeppelin ERC20 contract.

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We recommend using OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[FILLiquid Team, 2024/04/08]: `FIG` and `FIG` are self-defined tokens & transfer function is inherited from erc-20 standard, transfer function will always return true.

FIL-06 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/ERC20Pot.sol (base): 23, 37; contracts/FILGovernance.sol (base): 57, 66; contracts/FILTrust.sol (base): 44	● Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
23          _owner = owner;
```

- `owner` is not zero-checked before being used.

```
37          _owner = new_owner;
```

- `new_owner` is not zero-checked before being used.

```
57          _owner = new_owner;
```

- `new_owner` is not zero-checked before being used.

```
66          _burner = new_burner;
```

- `new_burner` is not zero-checked before being used.

```
44          _owner = new_owner;
```

- `new_owner` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

FIL-07 CHECK EFFECT INTERACTION PATTERN VIOLATED (OUT-OF-ORDER EVENTS)

Category	Severity	Location	Status
Concurrency	● Minor	contracts/ERC20Pot.sol (base): 32, 33; contracts/MultiSignFactory.sol (base): 237, 238	● Partially Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the reentrancy only causes out-of-order events.

External call(s)

```
32          _token.transfer(receiver, amount);
```

Events emitted after the call(s)

```
33          emit Transferred(receiver, amount);
```

External call(s)

```
 237          (bool success, bytes memory output) = info.target.call{value: msg.value
}(info.code);
```

Events emitted after the call(s)

```
 238          emit Executed(_msgSender(), proposalId, success, output);
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[CertiK, 2024/04/08]: The team partially resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f:

FIS-02 | MISSING CHECK ON `FILStake.setStakeParams()`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/FILStake.sol (base): 250~253	● Resolved

Description

In `FILStake.setStakeParams()` , there is no check preventing to set:

- `_minStakePeriod > _maxStakePeriod` ;
- `_minStake > _maxStakes` ;

Recommendation

We recommend enforcing `_minStakePeriod =< _maxStakePeriod` and `_minStake =< _maxStakes` to prevent unexpected errors.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

FIS-03 | `FILStake.setShares()` ALLOWS ZERO VALUES

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/FILStake.sol (base): 245~246	● Resolved

Description

The function `FILStake.setShares()` enforce that `new_rateBase == new_interest_share + new_stake_share` , however, this check does not prevent setting zero values as new parameters.

Recommendation

We recommend adding a check to ensure that only positive values are allowed.

Alleviation

[CertiK, 2024/04/11]: The team resolved the finding in commit b6ce507a20a2f517b66f3a3785aa0bc5dd543ef4

```
require(new_rateBase != 0 && new_interest_share != 0 && new_stake_share != 0 &&
new_rateBase == new_interest_share + new_stake_share, "factor invalid");
```

[CertiK, 2024/04/08]: The team partially resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f:

```
require(new_rateBase != 0 && new_rateBase == new_interest_share + new_stake_share,
"factor invalid");
```

only prevents the sum from being equal to zero, however, it is still possible for `new_interest_share` or `new_stake_share` to be zero when the other value is positive.

FIS-04 | CHECKS EFFECTS INTERACTION PATTERN NOT USED

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/FILStake.sol (base): 128~129, 158~162	● Partially Resolved

Description

In the function `unstakeFilTrust()`, the Checks-Effects-Interaction Pattern is not strictly followed. External Call ("Interaction") of token transfer in line 158 takes place before relevant state variables are updated.

Recommendation

Consider following the Checks-Effects-Interactions Pattern by putting the external call to `_tokenFILTrust` at the last line of the related functions.

Alleviation

[CertiK, 2024/04/08]: The team partially resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f:

- in `handleInterest()`, the external call takes place before the updates of the variables.

MSF-04 | MISSING THRESHOLD REQUIREMENT IN MULTISIG

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/MultiSignFactory.sol (base): 66, 98, 124	● Resolved

Description

Multisig wallet should always have a threshold of more than (at least) half of total signer count. In the `MultiSignFactory` , there is no enforcement of the requirement. The implication is that one multisig signer could potentially execute privileged functions without the agreement of a majority of signers.

Recommendation

We recommend enforcing the threshold requirement in the setup of the multi-sig wallet to be at least more than half of the signers count.

Alleviation

[CertiK, 2024/04/1]: The team resolved the finding in commit b6ce507a20a2f517b66f3a3785aa0bc5dd543ef4

FIL-08 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/FILGovernance.sol (base): 20~22, 24~26, 28~30, 40~42, 44~46, 56~59, 65~68; contracts/FILLiquid.sol (base): 805~807, 821~837, 923~940; contracts/FILStake.sol (base): 242~247, 249~254, 256~259, 273~285, 291~293; contracts/FILTrust.sol (base): 14~16, 18~20, 22~25, 27~29, 31~33, 43~46; contracts/Governance.sol (base): 356~388, 394~402, 408~411	● Partially Resolved

Description

It is important to emit events for sensitive actions, particularly those that can be executed by centralized roles or administrators. This ensures transparency and enables tracking of critical changes, which is essential for security and trust in the system. Missing event logs can result in a lack of visibility and potential information loss.

Recommendation

It is recommended to emit events in sensitive functions that are controlled by centralization roles.

Alleviation

[FILLiquid Team, 2024/04/08]: We cannot add events to contract `FILLiquid.sol` due to contract size limitation.

Commit: 575b30ec16bcb12c1ea6794511a888716749944f.

OPTIMIZATIONS | FILLIQUID

ID	Title	Category	Severity	Status
FIL-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Partially Resolved
FIS-01	If Statement Optimization	Coding Style	Optimization	● Resolved
MSF-01	User-Defined Getters	Gas Optimization	Optimization	● Resolved

FIL-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/ERC20Pot.sol (base): 18, 19, 20; contracts/FILLiquid.sol (base): 291, 292, 293, 299, 300, 318	● Partially Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[FILLiquid Team]: In FIL-01, modification advises are applied in contract `ERC20Pot.sol`. But in case of `Filliquid.sol`, such modifications could not be applied as it would increase the size of contract bytecode to such an extent that the contract would not be able to be deployed on chain.

Commit 575b30ec16bcb12c1ea6794511a888716749944f.

FIS-01 | IF STATEMENT OPTIMIZATION

Category	Severity	Location	Status
Coding Style	● Optimization	contracts/FILStake.sol (base): 143~144	● Resolved

Description

The variable `_onceStaked[staker]` only needs to be set to `true` if it is not `true` already, so line 144 can be included in the `if` statement for clarity and some gas savings.

Recommendation

Consider including line 144 in the `if` statement above instead.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

MSF-01 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/MultiSignFactory.sol (base): 111~113	● Resolved

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We advise that the linked variables are instead declared as `public` as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation

[CertiK, 2024/04/08]: The team heeded the advice and resolved the finding in commit 575b30ec16bcb12c1ea6794511a888716749944f.

FORMAL VERIFICATION | FILLIQUID

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-transfer-succeed-normal	`transfer` Succeeds on Valid Transfers
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transfer-never-return-false	`transfer` Never Returns `false`

Property Name	Title
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Valid Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract FILTrust (contracts/FILTrust.sol) In Commit 87d212ecce911e0e44a8df00bd82c3917cc5e261

Verification of ERC-20 Compliance

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-change-state	● True	
erc20-totalsupply-correct-value	● True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-false	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-revert-zero-argument	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-never-return-false	● True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-false	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-revert-zero	● True	
erc20-transfer-never-return-false	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-correct-amount	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	● True	
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-change-state	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-succeed-always	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-revert-zero	● True	
erc20-approve-correct-amount	● True	

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract FILGovernance (contracts/FILGovernance.sol) In Commit 87d212ecce911e0e44a8df00bd82c3917cc5e261

Verification of ERC-20 Compliance

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-change-state	● True	
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-never-return-false	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-revert-zero	● True	
erc20-transfer-false	● True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-zero-argument	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-change-state	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-succeed-always	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	● True	
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-false	● True	
erc20-approve-revert-zero	● True	
erc20-approve-correct-amount	● True	
erc20-approve-never-return-false	● True	

APPENDIX | FILLIQUID

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Financial Manipulation	Financial Manipulation findings indicate issues in design that may lead to financial losses.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract FILGovernance must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-change-state

The `totalSupply` function in contract FILTrust must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract FILTrust.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract FILGovernance.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender, msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender, msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                  && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false` .

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
requires recipient != address(0) && sender != address(0) && recipient != sender;
requires amount <= balanceOf(sender);
requires amount <= allowance(sender, msg.sender);
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result;
reverts_only_when false;
```

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `FILTrust` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-false

If the `transfer` function in contract `FILGovernance` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
requires recipient != address(0) && recipient != msg.sender;
requires amount <= balanceOf(msg.sender);
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result;
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

DISCLAIMER | CERTIK

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 you ("Customer" or 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, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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 CertiK 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 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.