Code Assessment
of the Franchiser V2 Smart Contracts

PUBLIC

Date July 02, 2026

Produced for Compound

By CHAINSECURITY

Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	6
5	System Considerations	8
6	Terminology	9
7	Findings	10
8	Limitations and use of report	13

1 Executive Summary

Dear Woof! team,

Thank you for trusting us to help Compound with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Franchiser V2 according to **Scope** to support you in forming an opinion on their security risks.

Compound implements the second version of Franchiser which allows the coordinator to reassign the COMP granted to franchiser pool without the need for a governance proposal. Moreover, it introduces a guardian who can freeze a franchiser pool and recall its assets in an emergency.

The most critical subjects covered in our audit are the trust boundaries of each role, how COMP moves between the different contracts of the system and whether it can be stolen. Only minor issues have been found. Security regarding all the aforementioned subjects is high.

The general subjects covered are gas efficiency of the implementation, the documentation and the specification. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Franchiser V2 repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Version	Date	Commit Hash	Note
1	22 June 2026	a33361d2ab2ef3465557f8abf27485357d076de3	Initial Review
2	2 July 2026	17b9a58da6af41d4651bb2f8051c862fb62b6b3d	Fixes

For the Solidity smart contracts, the compiler version `0.8.35` was chosen.

```
src/
├── Franchiser.sol
├── FranchiserPool.sol
└── FranchiserPoolFactory.sol
```

2.1.1 Excluded from scope

Any file not explicitly listed in the scope is assumed to be out of scope. In particular, third-party libraries like `openzeppelin-contracts` are out of the scope of this review and are assumed to work correctly and as expected. Additionally, `Franchiser` is not expected to be deployed and used as a standalone contract, the review focused on it being used as clones by `FranchiserPool`. The `Franchiser` is just a refactoring of the franchisers already deployed by the franchise factory. This means that its specification and implementation are assumed to be correct.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Compound is an evolution of the Franchiser system that gives the Compound DAO a more flexible way to manage the delegation of its COMP voting power. In the existing infrastructure, Governance must execute a dedicated proposal to fund a Franchiser, assign a delegatee, and delegate COMP atomically, and a second proposal to recall the voting power. As a consequence, Governance has to decide on the exact set of delegatees at proposal time, and any reassignment of voting power between delegatees requires a full proposal cycle. During that cycle, a delegatee whose voting power is about to be recalled can still vote, and a malicious or compromised delegatee may attempt to block the proposal.

To address this, the new system allows Governance to allocate a pool of COMP once and delegate the day-to-day distribution of voting power to a dedicated `Coordinator`, while a `Guardian` role can intervene in emergencies. The system consists of three components: the `FranchiserPoolFactory`, the `FranchiserPool`, and the `Franchiser`.

3.1 Components

A **Franchiser** is a contract that holds COMP and delegates the corresponding voting power to a single delegatee. The delegatee can in turn delegate part of that power to one further address (a sub-delegatee), and the owner can at any time recall the delegation from a delegatee together with its entire sub-delegate subtree. This is achieved by deploying child Franchisers owned by the parent Franchiser, forming a tree of Franchisers. This contract is reused unchanged from the existing implementation, including the limit of one sub-delegatee per Franchiser; only the surrounding infrastructure is new.

A **FranchiserPool** holds the idle COMP of a delegation program and is the entry point for the `Coordinator` and the `Guardian`. The `Coordinator` can delegate, recall, and reassign voting power among top-level delegatees, each managed through its own Franchiser. Idle COMP remains on the pool's balance, while delegated COMP is transferred to the top-level Franchisers. The number of active top-level delegatees is bounded by a configurable cap. The `Guardian` can recall specific delegatees or freeze the entire pool; while a pool is frozen, all `Coordinator` actions revert until a target block is reached or Governance lifts the freeze early.

The **FranchiserPoolFactory** is the entry point to the system for Governance (the Compound timelock). It deploys, funds, configures, and halts FranchiserPools, and is the only path through which Governance interacts with the pools. The Governance address is immutable.

4 Trust Model

We define the following trust model for the system.

4.1 Roles

Governance

- Trust level: Fully trusted. It is the Compound timelock, controlled by Compound governance, and its address is immutable in each pool.
- Permissions: Through the `FranchiserPoolFactory`, create and fund pools, halt a pool and move its entire COMP balance to an arbitrary recipient, replace the `Coordinator` and the `Guardian`, change the maximum number of delegatees and the freeze period, and unfreeze a pool early.
- Worst-case impact: Full control over the COMP allocated to the program. A malicious Governance could drain every pool to an arbitrary recipient via Halt Pool. This is out of scope, as Governance is the owner of the funds and is itself gated by the Compound governance process.

Guardian

- Trust level: Fully trusted. It is expected to be a community multisig that reacts in a timely manner to threats from a misbehaving `Coordinator` or from delegatees.
- Permissions: Recall specific delegatees back to the pool, recall all delegatees and freeze the entire pool, or suspend the `Coordinator` for the freeze period. Guardian actions cannot be frozen.
- Assumptions: It monitors the pool and acts before a malicious delegation can cause harm in a governance vote, and it coordinates with the `Coordinator` when recalling individual delegatees.
- Worst-case impact: It can never move COMP out of the pool or delegate voting power, so it cannot steal funds. A malicious or compromised Guardian can grief the program by repeatedly recalling delegatees and freezing the pool, denying voting power to all delegatees for up to the freeze period. Governance can replace it, and the Guardian cannot block that proposal because freezing does not remove Governance's ability to act.

Coordinator

- Trust level: Partially trusted by Governance. Expected to be a CGWG multisig.
- Permissions: When the pool is not frozen, delegate COMP from the pool to top-level delegatees, recall delegatees back to the pool, and reassign voting power between delegatees. It can never move COMP out of the pool.
- Worst-case impact: A malicious or compromised `Coordinator` can delegate voting power to addresses it controls and use the accumulated power to propose or vote for malicious proposals, up to the COMP held by the pool and the configured delegatee cap. It cannot extract COMP. In practice this is mitigated by the governance voting timeline: the `Guardian` has time to recall the voting power or freeze the pool, and Governance can replace the `Coordinator` while the pool is frozen, without the `Coordinator` being able to block that proposal.

Delegates and sub-delegates

- Trust level: Untrusted within the scope of the review.
- Permissions: A top-level delegatee receives voting power and may delegate part of it to a single sub-delegatee. They never have access to the underlying COMP.

- Worst-case impact: Beyond exercising the voting power explicitly granted to them, which could be used to support a malicious proposal if they accumulate enough, they cannot harm the system or access COMP. Their voting power can be recalled by the `Coordinator` or the `Guardian` at any time.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 `getAllPools` May Run Out of Gas

System Consideration | Version 1

`getAllPools()` returns the entire `_pools` set in a single call. As the number of pools grows unboundedly, the call may exceed the block gas limit and become unusable on-chain.

SC2 The Freeze Period Can Be Shortened

System Consideration | Version 1

Consider the following scenario:

- The guardian freezes a pool for time F
- At a point later in time the freeze period is set to F/2
- The guardian freezes the pool again.

In such a case, the freeze period will be shortened as freezing always succeeds. This is not an issue, but a property of the system that should be highlighted.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

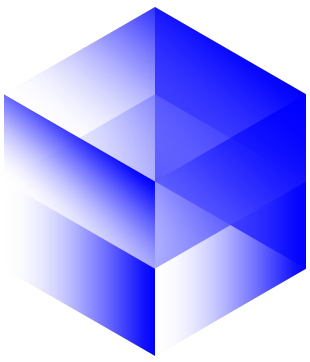
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

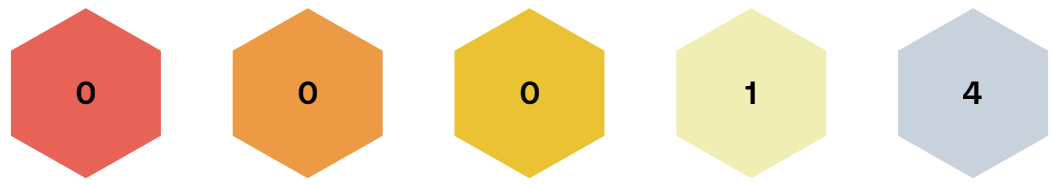
7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



No Open Issues

Total number of findings



Critical **High** **Medium** **Low** **Informational**

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Low	#001	Wrong Length Sanity Check in `emergencyRecallDelegates`	Code Corrected
Info	#002	`emergencyFreezePool` Does Not Recall Delegates	Specification Changed
Info	#003	`Franchiser` Implementation Token Not Validated	Code Corrected
Info	#004	Governance Unfreeze Can Revert	Acknowledged
Info	#005	Ownership Not Renounced in `Franchiser` Implementation Constructor	Acknowledged

7.2 Descriptions

#001 Wrong Length Sanity Check in `emergencyRecallDelegates`

Correctness	Low	Version 1	Code Corrected

`emergencyRecallDelegates` guards the input array with `delegatees.length >= maxDelegatees` instead of `>`. When the pool is full (the number of active delegatees equals `maxDelegatees`), the guardian can pass at most `maxDelegatees - 1` delegatees per call and can therefore never recall all of them in a single transaction. The condition is also stricter than necessary: it bounds the call by `maxDelegatees` rather than by the number of currently active delegatees. If `setMaxDelegatees` later lowers the cap below the active count, the check rejects every array large enough to recall the active set, leaving the function unable to recall all delegatees at all. Note that in the unlikely case where `maxDelegatees == 1`, the recall isn't possible and requires a governance proposal to be addressed.

Code Corrected

The check was removed as it's redundant.

#002 `emergencyFreezePool` Does Not Recall Delegates

Informational	Version 1	Specification Changed

The specification states that an emergency freeze must recall all delegates. `emergencyFreezePool` only calls `_freeze()` and leaves all delegatees active, while `emergencyFreezeAndRecallPool` matches the specified behavior. Having a freeze path that skips the recall contradicts the specification and leaves voting power distributed to delegatees during the freeze period.

Specification Changed

The specification has been updated to mention this behavior.

#003 `Franchiser` Implementation Token Not Validated

Informational	Version 1	Code Corrected

`FranchiserPoolFactory` hardcodes `votingToken` as the COMP constant but accepts `franchiserImplementation` as a constructor argument, checking only that it is non-zero. It never verifies that the implementation operates on the same token, i.e. that `Franchiser(franchiserImplementation).votingToken() == votingToken`.

Code Corrected

The factory now checks that the franchiser implementation uses the correct token.

#004 Governance Unfreeze Can Revert

Informational	Version 1	Acknowledged

Governance can unfreeze a pool through a proposal. Proposals typically bundle multiple actions, all of which are expected to succeed atomically. However, if the pool is already unfrozen by the time the proposal becomes executable, the unfreeze call reverts, which in turn causes the entire proposal to fail.

This is not an issue in itself, as the guardian can intervene and refreeze the pool so that the proposal succeeds. We highlight this property because the time at which a proposal becomes executable can vary depending on when quorum is reached, making it difficult to predict whether the pool will still be frozen at execution time.

Acknowledged

Compound responded:

> We acknowledge this behavior and are comfortable with it. We do not consider this issue significant enough to warrant changes.

#005 Ownership Not Renounced in `Franchiser` Implementation Constructor

Informational	Version 1	Acknowledged

The `Franchiser` constructor sets `msg.sender` as the owner of the implementation contract via `Ownable(msg.sender)`. Since the constructor runs only for the implementation (clones receive their owner through `initialize()`), the implementation is intentionally inert and should have no live owner. Retaining one leaves a privileged account with access to `recall` on a contract that is otherwise permanently locked.

Acknowledged

Compound responded:

> We acknowledge this observation; however, we do not see a scenario in which retaining ownership of the implementation contract could cause harm to the system. The implementation contract is intentionally inert and is not expected to hold any assets or be used directly.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.