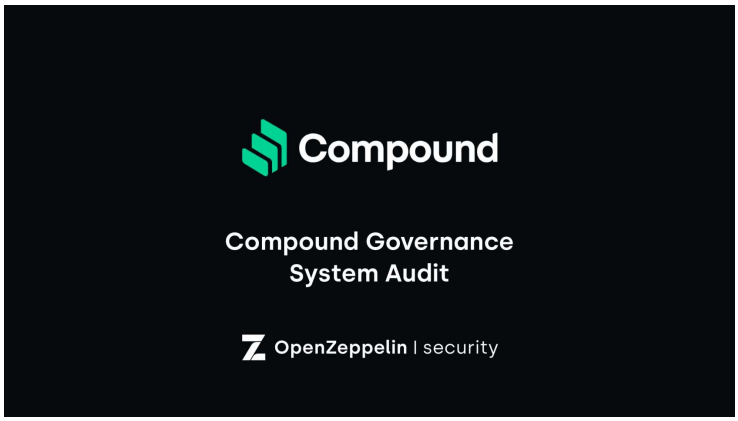
Compound Alpha Governance System Audit

FEBRUARY 25, 2020 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY



Compound Finance is a protocol deployed on the Ethereum network for automatic, permissionless, loans of Ether and various ERC20 tokens. It is one of the most widely used decentralized finance systems in the ecosystem.

Audit history and current scope

In this audit, we looked into Compound's alpha version of their governance system and its associated COMP token. While the initial audited commit is `6858417c91921208c0b3ff342b11065c09665b1b`, later the Compound team provided a follow-up commit `f5976a8a1dcf4e14e435e5581bade8ef6b5d38ea` which fixed some of the issues identified in the original commit. The scope included the newly introduced Compound Governance Token (COMP) and Governor Alpha contracts.

So far we have audited several phases of Compound's contracts. A brief summary of each phase is included

below.

1. A subset of Compound's contracts in commit `f385d71983ae5c5799faae9b2dfea43e5cf75262` of Compound's public repository. Read the report' summary.
2. A patch that introduced a time delay for critical admin functions and the ability to pause others. That patch is reflected in commit `681833a557a282fba5441b7d49edb05153bb28ec` of Compound's public repository. Read the report.
3. A refactor of the core `CToken` contract with the purpose of accommodating underlying tokens that may extract a fee when transferring tokens (e.g., USDT). This refactor is presented in commit `2535734126c7c26e9bc452f27f45c5408acff71f` of Compound's private repository. The report is not available to the public yet.
4. The difference between the code at commit `2535734126c7c26e9bc452f27f45c5408acff71f` of Compound's private repository and commit `bcf0bc7b00e289f9b661a0ae934626e018188040` of their public repository. These changes introduced the ability to handle underlying ERC20 tokens whose implementations can be upgraded (e.g., DAI). The report is not available to the public yet.
5. The difference between commit `bcf0bc7b00e289f9b661a0ae934626e018188040` and commit `9ea64ddd166a78b264ba8006f688880085eeed13` in Compound's public repository. The audit included changes to the `JumpRateModel` contract and the two newly added files `CDaiDelegate.sol` and `DAIInterestRateModel.sol`. The report is not available to the public yet.

High-level overview of the changes

The audited code change introduces a new governance system for Compound, including the Compound Governance Token (COMP) as well as an Alpha version of the core governance contract, respectively implemented in the audited `Comp` and `GovernorAlpha` contracts. The system should eventually fully replace Compound's administrator, now empowered to trigger sensitive modifications in the protocol via the a time lock mechanism in the `Timelock` contract. Since this is a preliminary version of the governance system, the `GovernorAlpha` contract is administered by a highly-privileged account, called `guardian`, with powers to:

- Cancel any non-executed proposal via the `cancel` function.
- Bypass the governance mechanism to change the `Timelock`'s admin at will (calling the `GovernorAlpha` contract `__queueSetTimelockPendingAdmin` and `__executeSetTimelockPendingAdmin` functions).
- This means that the Governor's guardian can override any queued proposal that attempts to change the `Timelock` contract's admin (i.e., a proposal that calls `setPendingAdmin`), as the guardian can call `__queueSetTimelockPendingAdmin` with whatever `eta` they decide.
- Make the `GovernorAlpha` contract accept administrative powers over the `Timelock` contract, calling the `__acceptAdmin` function.
- Abdicate, calling the `__abdicate` function.

The new administrator of the governance system is temporary, and is expected to abdicate once the system reaches a more stable stage.

The Compound Governance Token is intended to be a standard ERC20 token with extended functionality to allow token holders to delegate their votes, as well as

query an account's total amount of votes at a given block number.

The Compound Governor Alpha contract allows accounts that meet a certain vote threshold to submit proposals, which can then be voted on by delegated vote holders. Proposals with enough votes can be queued and executed in Compound's `Timelock` contract. Compound's governance system works closely with the `Timelock` contract to push proposal transactions into the `Timelock`, while being overseen by the guardian.

Following we present our findings, in order of importance.

Critical severity

None.

High severity

[H01] Approved proposal may be impossible to queue, cancel or execute

The `propose` function of the `GovernorAlpha` contract allows proposers to submit proposals with an unbounded amount of actions. Specifically, the function does not impose a hard cap on the number of elements in the arrays passed as parameters (i.e., `targets`, `values`, `signatures` and `calldatas`).

As a consequence, an approved proposal with a large number of actions can fail to be queued, canceled, or executed. This is due to the fact that the `queue`, `cancel` and `execute` functions iterate over the unbounded `targets` array of a proposal, which depending on the amount and type of actions, can lead to unexpected out-of-gas errors.

So as to avoid unexpected errors in approved proposals, consider setting a hard cap on the number of actions that they can include.

[H02] Queued proposal with repeated actions cannot be executed

The `GovernorAlpha` contract allows to propose and queue proposals with repeated actions. That is, two or more actions in a proposal can have the same set of `target`, `value`, `signature` and `data` values.

Assuming a proposal with repeated actions is approved by the governance system, then each action in the proposal will be queued individually in the `Timelock` contract via subsequent calls to its `queueTransaction` function. All queued actions are kept in the `queuedTransactions` mapping of the `Timelock` contract for future execution. While each action is identified by the `keccak256` hash of its `target`, `value`, `signature`, `data` and `eta` values, it must be noted that all actions in the same proposal share the same `eta`. As a consequence, repeated actions always produce the same identifier hash. So a single entry will be created for them in the `queuedTransactions` mapping.

When the time lock expires, the whole set of actions in a proposal can be executed atomically. In other words, the entire proposal must be aborted should one of its actions fail. To execute a proposal anyone can call the `execute` function of the `GovernorAlpha` contract. This will in turn call, for each action in the proposal, the `executeTransaction` function of the `Timelock` contract. Considering a proposal with duplicated actions, the first of them will be executed normally and its entry in the `queuedTransactions` mapping will be set to `false`. However, the second repeated action will share the same identifier hash as the first action. As a result, its execution will inevitably fail due to the

`require` statement in line 84 of `Timelock.sol`, thus reverting the execution of the entire proposal.

Consider modifying how each action in a proposal is identified so as to avoid clashes in their identifiers. This should allow for each action in a proposal to be identified uniquely, therefore enabling Compound's governance system to execute queued proposals that contain repeated actions.

Update: *Fixed in the follow-up commit* `f5976a8a1dcf4e14e435e5581bade8ef6b5d38ea` *which introduced a change to explicitly disallow proposals with repeated actions to be queued in the* `Timelock` *contract.*

Medium severity

[M01] GovernorAlpha contract does not fully match specification

- The `proposalApproved(uint256): bool` function mentioned in the specification is not implemented.
- According to the specification, a proposal can only succeed when, among other conditions, *"For votes are **greater** than the quorum threshold"*. However, in the implementation a proposal is considered successful when votes in favor are **equal or greater** than the quorum threshold.
- According to the specification, the `GovernorAlpha` contract should have a maximum number of operations that a proposal can contain. However, the audited implementation does not impose any limit on the number of actions (see `propose` function). This may allow proposers to submit proposals that may never be queued, canceled or executed (as explained in issue **[H01] Approved proposal may be impossible to queue, cancel or execute**).

Consider applying the necessary modifications to the code and / or to the specification so that they fully match. Should any deviation be intentional, consider explicitly documenting it with docstrings and inline comments.

[M02] Lack of allowance front-running mitigation in ERC20 token

The `Comp` contract is an ERC20 token contract that inherits from the `EIP20Interface` interface. However, it does not implement functions to mitigate the known ERC20 allowance front-running issue. This means that every token holder approving tokens to other accounts might be vulnerable to the front-running attack.

Consider implementing functions to safely increment and decrement approved amounts. For reference, see functions `increaseAllowance` and `decreaseAllowance` in OpenZeppelin's ERC20 de-facto standard implementation.

[M03] Proposal execution not handling returned data

The public `execute` function of the `GovernorAlpha` contract allows anyone to execute a queued proposal. Each action contained in the proposal will trigger a call to the `executeTransaction` function of the `Timelock` contract. The `executeTransaction` function returns a `bytes` value containing whatever data is returned by the call to the target address. It is important to note that the data is never logged in the emitted `ExecuteTransaction` event, thus it should be handled by the caller to avoid losing it. However, the returned data is not handled by the `execute` function of the `GovernorAlpha` contract. As a consequence, relevant data returned by the proposal's actions may be lost.

Consider handling the data returned by the subsequent calls to the `executeTransaction` function.

Potential courses of action to be analyzed include logging the data in events, or returning it to the `execute` function's caller in an array of `bytes` values.

Low severity

[L01] Lack of indexed parameters in events

None of the parameters in the events defined in the `GovernorAlpha` contract are indexed. Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.

[L02] Storage modification in require statement

Inside the `require` statement in line 143 of `Comp.sol`, the signatory's nonce is incremented right after being compared with the given nonce. In other words, the `require` statement fails if the given nonce is different from the one stored in the `nonces` mapping *before* it is incremented by one. Yet this subtlety of the language might no be caught by all readers, which can lead to confusions and errors in future changes to the code base.

To favor readability, consider incrementing the nonce outside the mentioned `require` statement, right after it has been verified.

[L03] Missing docstrings

All functions in the `GovernorAlpha` contract lack documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios

under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

[L04] Lack of input validation

- The `approve` function of the `Comp` contract does not ensure that the `spender` address is not zero. See OpenZeppelin's implementation for reference.
- The `propose` function of the `GovernorAlpha` contract allows the `description` parameter to be empty.

Consider implementing `require` statements where appropriate to validate all user-controlled input.

[L05] Not declaring return types in functions with return statements

Public functions `delegate` and `delegateBySig` of the `Comp` contract include a `return` statement at the end of their execution, although they do not explicitly declare return types in their definition. Moreover, both functions attempt to return the result of the internal `_delegate` function, which does not declare return types nor returns any value.

Consider removing the `return` statements of the `delegate` and `delegateBySig` functions, keeping in both cases the internal call to the `_delegate` function.

[L06] Undocumented, untested, custom behavior in transfer of ERC20 token

When the `transfer` and `transferFrom` functions of the `Comp` token are called, they internally call the `_transferTokens` function. This internal function can execute additional actions that are not part of the ERC20 standard. In particular, if the source and destination have different delegates registered, the `_decreaseVotes` and `_increaseVotes` functions are executed. This means that upon a transfer of tokens, delegates' votes amounts may be updated.

While the described custom behavior is fundamental to Compound's governance system, it was found to be undocumented and untested.

Consider explicitly explaining that delegates' votes can be updated in the docstrings of `transfer` and `transferFrom` functions. Furthermore, consider adding related unit tests in `CompTest.js` to ensure this sensitve feature works as expected.

Notes & Additional Information

[N01] Missing units

To avoid errors in future changes to the code base, consider using an inline comment to clearly state in which units the `votingDelay` is measured.

[N02] Not explicitly defining maximum allowance

To favor readability, consider declaring a constant `MAX_ALLOWANCE_AMOUNT` or `MAX_UINT256` to be used in the `transferFrom` function of the `Comp` contract instead of `uint(-1)`.

[N03] Declare uint as uint256

To favor explicitness, all instances of `uint` should be declared as `uint256`.

[N04] Inconsistent coding style

There are minor deviations from Compound's coding style. In particular:

- Public functions `__acceptAdmin`, `__abdicate`, `__queueSetTimelockPendingAdmin` and `__executeSetTimelockPendingAdmin` of the `GovernorAlpha` contract are unnecessarily prepended with double underscores.
- The `if` statement in line 234 of `GovernorAlpha.sol` is missing opening and closing braces.
- Internal functions `getChainId` in `GovernorAlpha` and `Comp` contracts should be prepended with an underscore to explicitly denote their visibility.

To favor readability, consider always following a consistent coding style throughout the entire code base.

[N05] Naming

- The `__acceptAdmin` function of the `GovernorAlpha` contract should be renamed to `__acceptTimelockAdmin`.
- The `NewProposal` event of the `GovernorAlpha` contract should be renamed to `ProposalCreated` to be consistent with other proposal-related events (e.g., `ProposalCanceled`, `ProposalQueued` and `ProposalExecuted`)

[N06] Typos

In the `Comp.sol` file:

- Line 33 should say `each account's` instead of `each accounts`.
- Lines 57 and 60 should say `that's` instead of `thats`.

- Line 57 should say `its delegate` instead of `their delegate`.

In the `README.md` file:

- `Copmtroller` should say `Comptroller` in the "Governor Alpha" subsection of the "Contracts" section.

[N07] Undocumented use of uint96 type

The `Comp` contract defines a `Checkpoint` struct to represent each checkpoint that marks an account's number of votes from a given block. This struct declares the number of votes as a `uint96` type to efficiently pack the struct data into 128 bits. Yet, this argument is currently not documented in the code. Note that the `uint96` type is also used in a few other places e.g. the `Receipt` struct of the `GovernorAlpha` contract, as well as the `balances`, `allowances` and `checkpoints` of the `Comp` contract.

Consider explicitly documenting the use of unusual Solidity types with inline comments to make the code more self-explanatory, thus favoring the project's readability.

[N08] Voting period assumes block frequency to calculate time

According to the `GovernanceAlpha` contract, the voting period is expected to last 17280 blocks, which given the current block time (around 15 seconds), would map to approximately 3 days. The number of blocks that the voting period lasts is currently hardcoded and cannot be modified by any means. However, it is known that Ethereum's "difficulty bomb" may increasingly make mining more difficult, thus increasing the average block time (see Etherscan's average block time for reference). As a consequence,

the voting period could eventually last much longer than expected.

Consider adding the necessary logic in the `GovernorAlpha` contract so that the voting period time may be adjusted via governance proposals if ever needed.

[N09] Redundant boolean check

Line 234 of `GovernorAlpha.sol` explicitly compares a boolean value to `true`. This is a redundant operation because the result will be equivalent to the boolean value itself. Consider removing the redundant comparison.

[N10] VoteCast event does not log the voter's address

Every time voters cast their votes for a proposal calling the `castVote` or `castVoteBySig` functions, a `VoteCast` event is emitted. However, this event does not currently log the voter's address, therefore hindering off-chain tracking of votes by voter.

Consider logging the address of the voter in the `VoteCast` event.

[N11] Contracts do not compile with solc 0.5.12

The `Comp` and `GovernorAlpha` contracts specify a compiler version equal or greater than solc 0.5.12. However, as seen in the output below, both contracts fail to compile with solc 0.5.12:

```
$ solc --version
solc, the solidity compiler commandline inter
face
Version: 0.5.12+commit.7709ece9.Linux.g++

$ solc --allow-paths . --evm-version istanbul
```

```
Governance/*.sol
Governance/Comp.sol:2:1: Warning: Experimenta
l features are turned on. Do not use experime
ntal features on live deployments.
pragma experimental ABIEncoderV2;
^------------------------------^
Governance/GovernorAlpha.sol:2:1: Warning: Ex
perimental features are turned on. Do not use
experimental features on live deployments.
pragma experimental ABIEncoderV2;
^------------------------------^
Governance/Comp.sol:282:20: Error: Variable n
ot found or variable not lvalue.
assembly { chainId := chainid() }
^-----^
Governance/GovernorAlpha.sol:299:20: Error: V
ariable not found or variable not lvalue.
assembly { chainId := chainid() }
^-----^
```

Consider only allowing compilation with versions greater than 0.5.12.

[N12] Inconsistent style for validating proposal state

The style for validating proposal state in the `cancel` function of the `GovernorAlpha` contract could be simplified to favor consistency with similar state validations in functions `execute` and `queue`. In particular, given that the `state` local variable is not used later in the `cancel` function, the `state` function can be called inside the `require` statement.

[N13] Incorrect error messages in require statements

Two `require` statements contain incorrect error messages. In particular:

- In line 258 of `GovernorAlpha.sol` : `castVote` should say `_castVote` .
- In line 261 of `GovernorAlpha.sol` : `castVote` should say `_castVote` .

Consider fixing these error messages to avoid confusions during debugging.

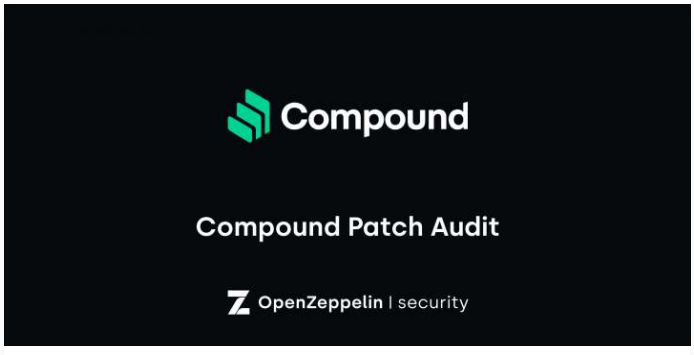
Conclusion

No critical and two high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Security Audits

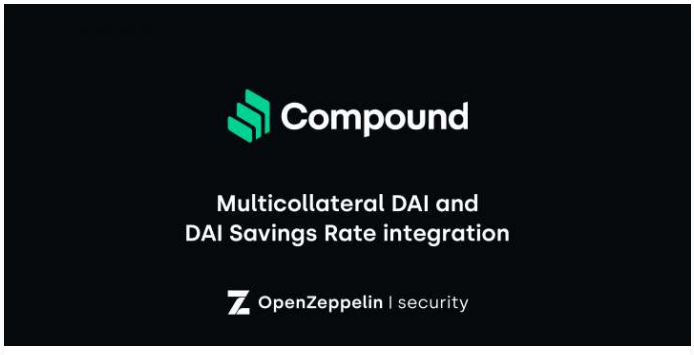
- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀
- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS



SECURITY AUDITS

Compound Finance – Timelock Audit



SECURITY AUDITS

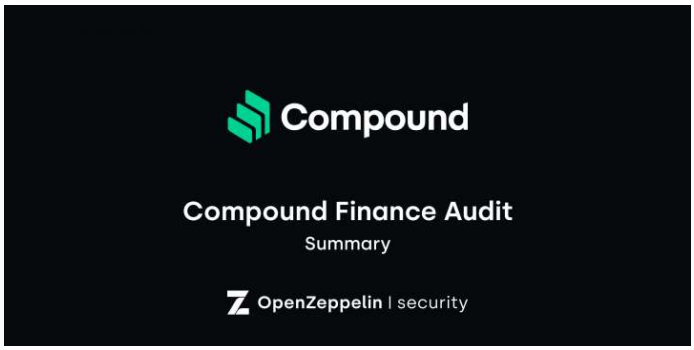
Compound Finance – MCD & DSR Integration

The Compound team asked us to audit a patch of their smart contract code. We examined the code and...

READ MORE

The Compound team asked us to audit their integration of Multi-Collateral Dai and the Dai Savings...

READ MORE



SECURITY AUDITS

Compound Finance Audit Summary

Compound Finance is a protocol, currently deployed on the Ethereum network, for automatic,...

READ MORE