Notional Contracts Audit

V2 Governance Contracts

OPENZEPPELIN SECURITY | NOVEMBER 5, 2021

Security Audits

The Notional Finance team asked us to review and audit the governance smart contracts of their v2 Protocol. We looked at the codebase and now publish our results.

Scope

We audited commit `c37c89c9729b830637558a09b6f22fc6a735da64` of the notional-finance/contracts-v2 repository. In scope were the governance contracts inside `contracts/external/governance/` directory:

- GovernorAlpha.sol
- NoteERC20.sol
- Reservoir.sol

We also reviewed the Constants.sol file to the extent it affected the governance contracts, as it is imported by the `GovernorAlpha.sol` file. However, the `Constants` library has not been reviewed in its entirety, nor all the files dependent on the `Constants.sol` file. In addition, the `NoteERC20` file imports `NotionalProxy.sol` but the proxy is not yet implemented and is thus out of scope for this audit. Moreover, there are placeholders in the contract which we make note of in our review where relevant. To ensure security, we recommend having the updated code audited once it is ready for review.

Note: the repository in question is private at the time of writing, so many hyperlinks will only work for the Notional Finance team.

System Overview

Notional is a protocol enabling fixed-term, fixed-rate lending, and borrowing on the Ethereum blockchain through a novel financial primitive named `fCash`, which provides a mechanism for users to commit to transfers of value at a specific point in the future.

The governance contracts specify an ERC20 token (i.e., `NOTE`) which is used for proposing upgrades and voting proposals. Votes represented by `NOTE` owners can be delegated to another address. Proposals have a minimum voting period and follow a timelock delay before being executed. Votes can be proposed by anyone with enough delegated `NOTE` tokens, and proposers must retain their `NOTE` balance until successful proposals are executed. A proposal in a non-executed state (include `Succeeded` and `Queued`) can be canceled by the `guardian` for any reason, or by any address if the proposer no longer has a sufficient delegated `NOTE` token balance.

There is another mechanism implemented by the `Reservoir` contract that can be funded by the governance and used to continually `drip` some token to a target until the reservoir balance is depleted. The contract contains `immutable` variables which are set in its `constructor` function. These may be deployed by governance at a later date.

Privileged roles

The Notional team will initially administer many aspects of the protocol. In particular, the `GovernorAlpha` contract defines a `guardian` address with powers to cancel proposals.

After an initial period, the team intends to use the `__abdicate` function to discontinue the use of a `guardian`.

Initially, the protocol will grant the `NOTE` tokens to certain addresses, and it is not known how many of these and how distributed those funds will be, funds that will count as voting power in future governance proposals.

best interest of the protocol. We assume that sensitive variables, such as `votingDelayBlocks` and `votingPeriodBlocks` are not set or updated to malicious values. We also discuss assumptions made about the unimplemented `getUnclaimedVotes` in the relevant findings below. We assume the deployment of `Reservoir` instances by governance is non-malicious.

Summary

Overall, we found the code to be clear to follow and read, with most functions and contracts properly documented. We highly recommend the team to improve the supporting documentation to describe every single contract and function. Moreover, there was a lack of comprehensive tests that could tackle edge scenarios and a coverage report.

Update: *The Notional team has addressed the issues identified in this report through the respective pull requests in the audited Github repository.*

Critical severity

None.

High severity

[H01] Proposal process could result in the wrong outcome

The `NoteERC20` contract keeps track of the voting power that users delegate in order to vote protocol's proposals.

The `getPriorVotes` function returns the number of votes an account had as of a given block using a binary search over all the available checkpoints. However, under certain cases when the pivot `center` checkpoint corresponds to the queried block number, meaning that `checkpoints[account][center].fromBlock == blockNumber`, only the `checkpoints[account][center].votes` value will be returned. Nevertheless, in all other instances the output will be the sum of the respective checkpoint votes and the value from the `getUnclaimedVotes` function.

return values in the binary search could be used as an attack vector to alter the result of the election. Consider updating the `getPriorVotes` function to handle `getUnclaimedVotes` consistently across all the outcomes.

Update: *Fixed in pull request 22.*

Medium severity

[M01] Approval process can be front-run

The ERC-20 standard `approve` function lets `NOTE` owners specify a `spender` that may transfer up to `rawAmount` tokens from the owner's balance.
The same `approve` function is used to make changes to this limit imposed on a `spender`, by calling the function again which replaces the value in the `allowances` mapping.

Performing a direct overwrite of the value in the `allowances` mapping is susceptible to front-running scenarios by an attacker (e.g., an approved `spender`). By monitoring the mempool for changes in the allowances, an attacker could spend both the previous and new `allowances` limits.

Although this vulnerability is acknowledged in a comment, it is not mitigated. Consider using the `safeIncreaseAllowance` and the `safeDecreaseAllowance` methods from the OpenZeppelin's `SafeERC20` library.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> _Won't fix, unclear how one would resolve this in the "approve" method without adding the separate atomic increaseAllowance and decreaseAllowance methods. My feeling on this issue is that it has to be dealt with client side anyway (increaseAllowance / decreaseAllowance are non-standard) and if the client is savvy enough to be aware of those methods it can also enforce that the allowance is set to zero before it is increased. Also just from a practical perspective, allowances are generally set to MAX*UINT256 or 0 for most apps anyway just to simplify the user experience.*

addresses and `initialGrantAmount` array of values to assign the grant tokens into their accounts. In the loop through the `initialGrantAmount` array, all the grants are accumulated in the `totalGrants` variable, and the `initialAccounts` accounts are granted their corresponding `initialGrantAmount` amount by updating their balances.

However, if the deployment transaction has duplicated elements in the `initialAccounts` array, the `totalGrants` will represent the sum of all the `initialGrantAmount` elements, but the balances for those accounts will not reflect the sum of all their grants. This is due to the overwrite operation that it is made to all the balances which will remove all old balances of that duplicated account.

Although this issue could be mitigated by redeploying the contract while bootstrapping the protocol, the effect of the issue could be more serious in cases where the governance is in charge of updating the `NoteERC20` contract through proposals. It could become impossible to vote new proposals or fix the protocol if the available voting power is not enough to pass a new proposal. Consider validating that there are not duplicate values in the `initialAccounts` array.

Update: *Fixed in pull request 23 by checking if the balance is zero before granting tokens to the initial accounts.*

[M03] Lack of event emission after sensitive actions

The following functions do not emit relevant events after executing sensitive actions.

- In the `GovernorAlpha` contract, the `__abdicate` function does not emit any event when the `guardian` renounces its role.
- In the `Reservoir` contract, the `drip` function does not emit a distinct event besides a regular `Transfer` event.

Consider emitting events after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Fixed in pull request 25.*

[M04] Incomplete set of unit tests

no way to determine if the current implementation matches the system's expected behavior. Thorough unit tests aiming at 95% coverage are in order for the security of the project to be assessed in a future audit. Integrating test coverage reports in every single pull request of the project is also highly advisable.

Update: *Partially fixed in pull request 33. The test file no longer has empty units of tests, although no further cases have been tested.*

[M05] Test coverage script fails to deliver the report

Even though the project uses the `Brownie` package to automate the tests and compile the contracts, running the script to get the coverage of the tests throughout the codebase ends up in a frozen task without ever delivering the actual code coverage report. Without this report it is impossible to know whether there are parts of the code never executed by the automated tests; so for every change, a full manual test suite has to be executed to make sure that nothing is broken or misbehaving.

Consider fixing and documenting how to run the test coverage scripts and making that coverage reach at least 95% of the source code.

Update: *Acknowledged. Notional's statement for this issue:*

> *This is currently an issue with performance limitations in ganache and the way that brownie runs its tests. It's on my list to dig into this in more depth and get it fixed but won't be fixed as a result of this audit.*

[M06] `Reservoir` does not accept `ETH`

The `Reservoir` contract has no way to accept `ETH`, even though governance can execute proposals that handle `ETH`.

Whether the `Reservoir` contract should handle `ETH` or not, for instance by using `WETH` instead of `ETH`, this expectation should be properly documented. Consider either implementing the functionality to allow the `Rerservoir` contract to handle and drip `ETH` or documenting the reasons behind this choice.

The `Reservoir` contract allows the protocol to distribute a token at a fixed rate, predominantly to the protocol's reserve, by dripping them every so often.

The parameters involved in the process are set during the deployment of the contract and it is not possible to modify them afterwards.

Because this contract may be deployed several times to drip a variety of tokens at the same time, it is possible that during the process a certain `tokenA` could be sent by mistake to a `Reservoir` contract that only handles a `tokenB`. In such scenario, all the `tokenA` funds sent to that contract will get stuck due to the impossibility of withdrawing/transferring them to any other address.

Consider implementing the functionality to withdraw or transfer those tokens, different from the `TOKEN` token, to the reserve to prevent that funds could get trapped inside the `Reservoir` contract.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> *Won't fix, the intention is to only use the reservoir for NOTE tokens.*

[M08] Untested custom SafeMath library in use

The `GovernorAlpha` and the `NoteERC20` contracts make use of a `SafeMath` alike library embedded into the contracts based on OpenZeppelin's `SafeMath` library, customized to support safe mathematical operations for unsigned integers of 32 and 96 bits. Yet, none of the functions in the implemented `SafeMath` library are tested, which may hinder users and developers' trust in this custom library. Moreover, any change in the library is not going to be detected by the current test suite, rendering all business logic depending on it vulnerable to potential security issues introduced by these future modifications.

Consider including thorough and extensive unit tests for the `SafeMath` library.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> *Won't fix, have manually tested that the methods don't overflow.*

the initial accounts.

However, the index `i` used to cycle among all the elements is not initialized with zero and its default value is used instead. Relying on the default value of a variable is not a recommended and secure methodology to perform due to the unexpected effects that may occur when this is not true, whether it is the result of a compilation bug or the outcome of a security issue.

Similarly, during the contract initialization, the `totalGrants` variable is used as a counter for all the distributed grant tokens but it is not initialized with a zero value before its usage.

Consider initializing all variables, specially indexes and counters, that will use their initial value during operation.

Update: *Fixed in pull request 23.*

Low severity

[L01] Users can emit `Approval` and `VoteCast` events to deceive offline services

The `NoteERC20` contract implements the functionalities of the `NOTE` token that allow users to delegate their voting power and vote proposals. However, if a non-allowed user tries to transfer zero number of tokens on behalf of another another account by calling the `transferFrom` function, the transaction will not only succeed but will also emit both the `Transfer` and `Approval` events, allowing a malicious user to disguise this harmless transaction as an approval on a non-owned account.

Similarly, in the `GovernorAlpha` contract, the `_castVote` function can also be successfully used by an user that does not have any voting power to submit their vote, resulting in the emission of the `VoteCast` event.

Although these situations will not impact in the voting's result nor the users' balances, the emission of these `Approval` and `VoteCast` events may be used to spam offline services, to confuse users, or to perform phishing attacks. Consider either requiring non-zero values in those functions

Update: *Fixed on pull request 26. However, the changes has broken the* `ERC20` *compliant token compatibility because no* `Transfer` *event is emitted on zero value transfers. Consider emitting the* `Transfer` *event on such cases.*

[L02] Lack of guardian-role transfer

The guardian address from the `GovernorAlpha` contract has a special role inside the system.

However, if the address associated with the `guardian` gets compromised, there is no way to transfer those special roles to another address.

Consider creating a function to transfer the `guardian` role to another address or consider documenting this limitation.

Update: *Fixed in pull request 30.*

[L03] Implicit casting

Throughout the codebase, many instances of implicit casting between types exist.

Some examples of such behavior can be found in the `NoteERC20.sol` file on:

- Line 96 where the each element of the `initialGrantAmount` is being casted in the `Transfer` event.
- Line 99 where an implicit casting on the granted amount occurs when it is being compared to the total supply.
- Line 107 where the allowance in `uint96` for a pair of accounts is casted into `uint256` by the return operation.
- Line 127 where the amount value is casted in the `Approval` event.
- Line 135 where the balance in `uint96` is casted into `uint256` by the return operation.

Whenever a different type of variable is needed, consider either checking and casting the variable into the desired type or using OpenZeppelin's `SafeCast` library which provides overflow checking when casting from one type of number to another.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

[L04] Misleading comments

Throughout the codebase there are cases of misleading or inaccurate documentation. Some examples are:

- The description of the `dripRate_` parameter for the `Reservoir` `constructor` claims to be “tokens per block”, but the implementation is actually “tokens per second” as described in the `DRIP_RATE` variable.
- The documentation for the `quorumVotes` parameter state that “for a vote to succeed” although it should be either “for a voting process to succeed” or “for a proposal to succeed”.

Consider fixing these and all misleading comments in the codebase to improve the code’s readability.

Update: *Fixed in pull request 27.*

[L05] Lack of input validation

Several `external` functions fail to validate the input parameters supplied. For example:

- The initialization of the `Reservoir` contract completely lacks input validation, such as validating whether the `token_` address corresponds to a real contract and whether the `dripRate_` value is greater than zero.
- The `notionalProxy_` address from the `NoteERC20` contract does not check if that address corresponds to a contract’s address.
- The `constructor` function from the `GovernorAlpha` contract does not check if the `note_` address corresponds to a contract’s address. Moreover, the `quorumVotes_`, the `proposalThreshold_`, the `votingDelayBlocks_`, and the `votingPeriodBlocks_` values are not being check in any sense, which could result in the ability to vote a proposal without the need of voting power, the inability to vote a proposal if the voting period is zero, or the inability to pass a proposal due to a shortage of needed voting power, among other outcomes.
- Similarly, in the `GovernorAlpha` contract, the `updateQuorumVotes`, the `updateProposalThreshold`, the `updateVotingDelayBlocks`, and the

To avoid errors and unexpected system behavior, consider explicitly restricting the range of inputs that can be accepted for all externally-provided inputs via `require` clauses where appropriate. For all cases of contract's addresses, consider using the OpenZeppelin's `Address` library to check that the input address corresponds to a contract's address.

Update: *Partially fixed in pull request 32. Some of the protocol's parameters are still not checked against a zero value. Furthermore, the code does not enforce validation over the voting period due to conflicts with the tests. Notional's statement for this issue:*

> *No validation is done for quorumVotes, proposalThreshold and votingDelayBlocks because it is not clear what values would be reasonable minimums or maximums there.*

[L06] Missing and / or incomplete docstrings

Some contracts and functions in the code base lack documentation or include incomplete descriptions. 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. Below we list all instances detected during the audit.

In the `Constants` library:

- Several constants lack their documentation stating its purpose and the reason of its value, such as the `FCASH_ASSET_TYPE` constant.

In the `GovernorAlpha` contract:

- The `getReceipt` function is missing a docstring for the `return` value.
- The `state` function contains an incomplete docstring for the `return` value.
- The `updateQuorumVotes` function is missing all its documentation.
- The `updateProposalThreshold` function is missing all its documentation.
- The `updateVotingDelayBlocks` function is missing all its documentation.
- The `updateVotingPeriodBlocks` function is missing all its documentation.
- The `constructor` function is missing all its documentation.

documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially fixed in pull request 27. No documentation has been added to the* `Constants` *library, such as on the* `FCASH_ASSET_TYPE` *constant.*

[L07] Not using SafeMath functions

Although several parts of the codebase employs its custom SafeMath methods where appropriate, there are still a few instances of regular Solidity arithmetic operators being used. Some examples are:

- On line 191 of `GovernorAlpha.sol` a `+` is used.
- On line 56 of `Reservoir.sol` a `-` is used.
- On line 221 of `NoteERC20.sol` a `-` is used.
- On line 244 of `NoteERC20.sol` a `-` is used.
- On line 245 of `NoteERC20.sol` a `-` is used.
- On line 254 of `NoteERC20.sol` a `-` is used.
- On line 361 of `NoteERC20.sol` a `-` is used.
- On line 365 of `NoteERC20.sol` a `+` is used.

Although these cases do not represent a security risk per se, consider always performing arithmetic operations with methods that protect the system from such possibilities, like the math libraries of OpenZeppelin contracts.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> *Won't fix, overflow in any of these cases is highly unlikely*

[L08] Transfer of ERC20 tokens do not use SafeERC20

The `drip` function from the `Reservoir` contract transfers a fix-rate of ERC-20 tokens to a target address based on its deployment parameters. The `transfer` call used is then checked for success. However, some ERC-20 compliant tokens may not return a boolean.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> *Won't fix, there's no intention of expanding the use case beyond NOTE tokens*

[L09] Proposals never reach the `Queued` state

The `state` function from the `GovernorAlpha` contract is used to return the current state of any proposal. The possible states are stored in the `ProposalState` enum and the `proposals` mapping contains a record of all proposals data in any state.

Due to the logic used for determining the `Succeeded` state, a queued proposal will never be able to reach the `Queued` state, even after the `queueProposal` function has been called.

Although the `state` function is not currently used to determine whether a proposal is `Queued` or not, changes to this, other contracts, or any other project that rely on the `ProposalState` being `Queued` may be vulnerable to future attacks. Consider revising the way the `state` function identifies a proposal as `Queued` and warning users about this behavior.

Update: *Fixed in pull request 31. The `state` function now validates if the proposal is queued by checking if it is still pending.*

[L10] Re-implementing ECDSA signature recovery

The `castVoteBySig` function from the `GovernorAlpha` contract and the `delegateBySig` function from the `NoteERC20` contract include an implementation of the ECDSA signature recovery function. This function is already part of the OpenZeppelin contracts, which has been audited and is constantly reviewed by the community.

Consider importing and using the `recover` function from OpenZeppelin's ECDSA library not only to benefit from bug fixes to be applied in future releases, but also to reduce the code's attack surface.

Update: *Fixed in pull request 28. However, because the OpenZeppelin's `ECDSA` library already checks if the outcome is the zero address, the same validation from the `GovernorAlpha`*

[L11] Proposal execution not handling returned data

The `executeProposal` function from the `GovernorAlpha` contract allows any user to execute a proposal from the timelock queue once the delay has passed. This is accomplished through the `executeBatch` function from OpenZeppelin's `TimelockController` contract.

Currently the `executeBatch` function does not handle data returned by executed transactions. Although OpenZeppelin plans to implement this soon in the `TimelockController` contract, consider extending the functionality in order to handle return data from the proposals' execution.

Update: *Acknowledged. Notional's statement for this issue:*

> *Won't fix, will use the OZ implementation when it is ready. It's not clear how a set of governance proposals will actually use return data during a batch of transaction executions.*

[L12] Unnecessary `return` statements

The `delegate` and `delegateBySig` functions use `return` when calling the `_delegate` function. However, because the `_delegate` function does not return any output, these return statements are needless.

To avoid confusion and improve the code's readability, consider removing the unnecessary `return` statements from the code.

Update: *Fixed in pull request 28.*

Notes & Additional Information

[N01] Event parameters are not indexed

Throughout the codebase, there was a lack of index used on event parameters. For example, the `VoteCast` and `ProposalQueued` event parameters are not indexed. Consider indexing event parameters so event logs are easier to be queried and used by offline services.

Update: *Fixed in pull request 29.*

enact all the transactions corresponding to a proposal by calling the `executeBatch` function from the OpenZeppelin's `TimelockController` contract through the `_executeBatch` function. However, if the proposal is still queued, the transaction will revert just before the end while checking the timelock, instead of failing early before several calls to contracts are made.

Consider reviewing the codebase for such occurrences and refactoring by returning or failing earlier for gas efficiency.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> *Won't fix, will just rely on the client to check that this works before submitting.*

[N03] Inconsistent coding style

Some instances of inconsistent coding style and style guide deviations were identified in the code base. Specifically:

- The `GovernorAlpha` contract inherits the functionality of the OpenZeppelin's `TimeLockController` contract. If a proposal is cancelled, the contract will trigger two events as shown in the documentation. However, the names of the events are spelled in different English styles. To improve the codebase's consistency, the name of the event from the `GovernorAlpha` contract should be changed to match its equivalent event in the `TimeLockController` contract.
- The functions in the `GovernorAlpha` contract do not follow the recommended order of: constructor, fallback, external, public, internal, private. For example, the `private` `_computeHash` function comes before the `external` `queueProposal` function.
- The `GovernorAlpha` contract checks if the current block number fits inside a `uint32` variable type in order to prevent errors due to its internal accounting system. The `NoteERC20` contract also performs this check by using a dedicated function in several places of the code. However, the `GovernorAlpha` contract does not perform this check in all places where the block number is used, such as in the `cancelProposal` or the `state` functions.
- `Constants` should follow the UPPER_CASE notation. Some deviations were identified, such as the `name` and `proposalMaxOperations` `constants`.

For functions with lots of arguments, such as [redacted], consider dropping each argument onto it's own line.

- The `GovernorAlpha.sol` file implements a minimal interface for the `NoteERC20` contract which is then used by the `GovernorAlpha` contract. However, the `NoteInterface` name does not follow the usual naming style for interfaces, which should be `INoteERC20`. Furthermore, this interface is placed inside the same file where the contract that uses it is, instead of importing the respective interface's file at the beginning of the file.
- Some revert messages in the `GovernorAlpha` contract do not follow the same style as other revert messages in the same contract.
- GoverorAlpha provides `""` as the optional `predecessor` argument for `hashOperationBatch`, instead of the recommended `bytes32(0)`.

To favor readability, consider always following a consistent style throughout the code base. We suggest using Solidity's Style Guide as a reference with the help of linter tools such as Solhint.

Update: *Partially fixed on pull request 29. Issue has been updated based on the team's response. Further Notional's statements for this issue:*

> *Won't fix, prefer to have internal / private methods near the functions where they are referenced for readability*
>
> *Won't fix `name` as it is used as a public getter as well*
>
> *Won't fix, not compatible with with prettier config*
>
> *Won't fix revert messages*

[N04] Proposal states overlap during voting expiry

The `GovernorAlpha` contract implements the `state` function that outputs the current state of a proposal.

However, there are two states that overlap their conditions during the expiration of the voting period: the `Active` and the `Succeeded` states.

Consider restricting the `Succeeded` state to be active when the block number is strictly greater than the `endBlock` value.

Update: *Fixed in pull request 29.*

[N05] Revert messages statements are missing

Some `require` statements are missing their respective revert messages. For example:

- line 160 in `GovernorAlpha.sol`.
- line 88 in `NoteERC20.sol`.
- line 93 in `NoteERC20.sol`.
- line 99 in `NoteERC20.sol`.

To improve the code's readability and to help debugging issues that may arise, consider always including revert messages in all `require` statements.

Update: *Acknowledged, and will not fix. Notional's statement for this issue:*

> *Won't fix, require strings add to deployed code size which increase gas costs for users. Initialize will only be called once so this is a fair trade off.*

[N06] Unused import

The `GovernorAlpha` contract imports `Constants.sol`, but does not use it. To improve readability and avoid confusion, consider removing unused imports.

Update: *Fixed in pull request 29.*

Conclusions

1 high severity issue was found. Several changes and recommendations were proposed to reduce the code's attack surface and improve its overall quality.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits