OpenZeppelin | security

Venus Multichain Governance Audit

January 19, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-12-05 To 2023-12-19
Languages	Solidity

Total Issues	22 (22 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	3 (3 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	5 (5 resolved)
Notes & Additional Information	12 (12 resolved)

Scope

We audited the VenusProtocol/governance-contracts repository at commit 7304a68, except for `TimelockV8.sol`, which was audited at commit 8a3d31f.

In scope were the following contracts:

```
contracts
├── Cross-chain
│   ├── BaseOmnichainControllerDest.sol
│   ├── BaseOmnichainControllerSrc.sol
│   ├── interfaces
│   │   ├── IGovernananceBravoDelegate.sol
│   │   └── ITimelock.sol
│   ├── OmnichainExecutorOwner.sol
│   ├── OmnichainGovernanceExecutor.sol
│   └── OmnichainProposalSender.sol
└── Governance
    └── TimelockV8.sol
```

System Overview

The Venus protocol will be deployed to more networks. As such, the multichain governance contracts are designed to allow proposal submission and execution on the new networks using the current Venus governance system, whereby XVS holders can vote for/against the proposals. The governance system on the BNB Chain is extended with a proposal sender (`OmnichainProposalSender`) which can send proposals to destination chains via LayerZero endpoints.

Proposal executor contracts (`OmnichainProposalExecutor`) are deployed on the destination chains to receive messages via LayerZero which contain transaction payloads as part of a Venus Improvement Proposal (VIP). These payloads enter a timelock which can be executed later based on a critical, fast-track, or normal priority. The sender and executor contracts use the same base logic as that which is provided in LayerZero's omnichain governance executor example contracts.

The messaging behavior is non-blocking, meaning that any failed messages (e.g., not providing enough gas payment for execution on the destination chain) will not prevent future messages from being sent. Instead, the failed message payloads are stored locally to retry sending them in the future. In addition, both sender and executor contracts impose a max daily limit on the number of commands that can be sent/received within a 24-hour window.

Security Model and Trust Assumptions

The main trust requirement is that of the LayerZero endpoint. The endpoint receives all proposal payloads from the sending contract on the BNB chain, and is responsible for forwarding the payloads in their original state to the correct addresses on the destination chain. A compromised endpoint could allow, for example, malicious payloads to be sent to executor contracts resulting in arbitrary state changes in the Venus contracts. To mitigate this possibility, Venus imposes a two-sided daily command limit on the sender and executor contracts. This is in addition to timelock delays on the executor side before anyone is able to execute the received payloads.

Privileged Roles

Those authorized in the ACM can perform the following privileged actions:

- Pause and resume sending/receiving messages
- Set max daily command limits on the sender
- Send proposal commands by calling `execute` in `OmnichainProposalSender`
- Set trusted remote addresses (destination chain receiving addresses)
- Configure parameters of the LayerZero endpoint

The owner can perform the following privileged actions:

- Set the timelock addresses in `OmnichainProposalExecutor`
- Set max daily command limits on the executor
- Set valid signatures of functions that can be called via `OmnichainExecutorOwner`

The `guardian` address configured in any executor contract has the ability to cancel any unexecuted proposals.

High Severity

H-01 Receiver Will Fail to Queue if Same Transaction Exists Multiple Times in One Proposal or Exists In Multiple Proposals

When queuing and executing, the `TimelockV8` contract tracks transactions using a mapping with hashed values:

```
bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta))
```

Since `eta` is constructed as blocktime + delay of this timelock, if identical transactions are queued at the same time, the queuing will revert. Having identical transactions in the same proposal is not uncommon, especially in complicated governance proposals (e.g., switching a storage flag before or after an action).

Consider thoroughly documenting the design intention and add necessary checks if the intention is to not allow identical transactions to be queued at the same time. Otherwise, consider modifying the logic to ensure that the timelock hash values cannot collide.

Update: *Resolved. The Venus team stated:*

> *Since we have similar check in governanceBravoDelegate to ensure that proposal should not contain identical transaction. https://github.com/VenusProtocol/governance-contracts/blob/develop/contracts/Governance/GovernorBravoDelegate.sol#L295*

H-02 Earlier Proposals Cannot Be Executed Until Proposal Count Is Increased

When a message to execute a remote proposal is sent, its `pId` is obtained from `governanceBravoDelegate.proposalCount` and is stored as the last proposal id in the proposal sender contract. However, any subsequent attempts to execute a remote proposal will fail if the proposal count is less than or equal to the last proposal id.

This means that if there are multiple proposals proposed in `governanceBravoDelegate`, only one can be executed. A new proposal would have to be added to increase the proposal count just to execute the earlier proposals. Consequently, it is never possible to execute all outstanding proposals if multiple proposals exist.

Consider using different logic to ensure the uniqueness of remote commands.

Update: *Resolved at commit 1f44629, pull request #54.*

H-03 Gas Could Get Stuck on Source Chain

When sending a proposal across chains, `OmnichainProposalSender` takes a source chain gas payment from the original caller in the `execute` function (as `msg.value`) and passes this to the `lzEndpoint` in a `try catch` block at line 153. However, it does not have a refund mechanism if the call to `lzEndpoint.send()` fails for any unexpected reason other than running out of gas. As such, all unused gas will be returned to `OmnichainProposalSender` instead of the original caller. There is no way for the original caller to withdraw this gas payment and it will be stuck in the contract forever.

Consider refunding the gas payment from `msg.value` to the caller in the `execute` catch block if the call fails for a reason other than running out of gas.

Update: *Resolved at commit 9029c19, pull request #54.*

Medium Severity

M-01 Transactions Might Fail to Be Queued in Timelock if Admin Reduces the Timelock Delay Time

`TimelockV8` allows the governance to reset `delay` length through the `setDelay` function. However, there are no restrictions on the proposed new delay value. This could lead to a collision with an existing queued transaction in the timelock if the following conditions are met:

- Old and new transactions are identical but were queued at different times
- The time difference between these two transactions matches the reduced amount time of the old delay value

For example,

1. Transaction 1 was queued at blocktime 0 with a delay of 2 days
2. The delay was updated to 1 day
3. An identical transaction is queued at blocktime 1 day with a delay of 1 day

This will lead to the same `eta` for the two identical transactions, leading to a collision in the timelock mapping of transactions and causing the second transaction queuing to revert.

Consider either only allowing increasing the delay value or ensuring that there is no collision with existing queued transactions before reducing it.

Update: *Resolved. The Venus team stated:*

> *Since function setDelay(uint256 delay_) can only be executable via VIP, the situation outlined in the finding can be addressed when executing a proposal from source chain. We will make sure that no proposal that is identical to the one that comes after the delay is reset is in the queue. Moreover this situation is very unlikely to happen.*

M-02 Governance Might Not Be Able to Send Urgent Proposals due to Failed Proposals Taking Up Daily Sending Limit

When sending proposals across chains, a 24-hour `maxDailyLimit` quota is enforced on the destination chainId. This is done through the `_isEligibleToSend` function. However, if a proposal has failed to be sent, the transactions in the proposal still occupy the daily limit because `execute()` in `OmnichainProposalSender` does not reset the limit inside the catch block at line 163.

This could lead to a scenario whereby urgent proposals are unable to be sent out if there are other successful and failed proposals within the last 24-hour window filling the limit. It could also cause the reverse scenario whereby transactions exceeding the limit are sent to the destination chain due to some failed proposal being re-executed.

Consider resetting the limit for any proposals that failed to be sent.

Update: *Resolved. The Venus team stated:*

> *The most frequent mistake that causes a proposal to revert its bridging is running out of gas. In the scenario described in the findings, we will take the following actions to make sure that no urgent proposal gets stuck because the limits have been exceeded: 1.*

> *Increase the maximum daily limit via a CRITICAL proposal for a source and destination chain. 2. The next proposal would be the urgent proposal.*

Low Severity

L-01 Incorrect Documentation of Payload Parameters

The payload is documented as being an encoding of targets, values, signatures, and calldatas. However, when the payload is decoded, it expects an additional encoded parameter, `pType` of type `uint8`, which is not documented anywhere in the code with respect to the payload (although it is expected to be the proposalType).

Consider adding clear documentation describing each parameter of the payload accurately.

Update: *Resolved at commit 4b6b78c, pull request #54.*

L-02 Lack of Input Check

The `execute()` function of `OmnichainProposalSender` does not check whether the input `payload_` is empty or not.

Consider adding this check to ensure that the correct input value is provided.

Update: *Resolved at commit 7c50b32, pull request #54.*

L-03 Guardian Can Cancel Non-Queued Proposals

The `cancel` function of `OmnichainGovernanceExecutor` allows the guardian to cancel a queued proposal from the corresponding timelock. However, this process does not check whether the proposal has been queued yet. It is possible for the guardian to cancel a non-queued proposal, leading to incorrect states of `proposal.cancelled` and event pollution.

Consider checking whether a proposal is queued or not before allowing the guardian to cancel it.

Update: *Resolved at commit f018826, pull request #54.*

L-04 Lack of `gap` Variable

The upgradeable `OmnichainExecutorOwner` contract does not have a `gap` variable.

Consider adding a `gap` variable to avoid future storage clashes in upgradeable contracts.

Update: *Resolved. The Venus team stated:*

> *Not required as of now, OmnichainExecutorOwner is not inherited by any contract. The current storage layout would be AccessControlledV8 Storage followed by OmnichainExecutorOwner storage.*

L-05 Unsafe ABI Encoding

It is not uncommon to use `abi.encodeWithSignature` or `abi.encodeWithSelector` to generate calldata for a low-level call. However, the first option is not typo-safe and the second option is not type-safe. Thus, both of these methods are error-prone and should be considered unsafe.

At line 198 of `OmnichainGovernanceExecutor.sol`, an unsafe ABI encoding is used.

Consider replacing all instances of unsafe ABI encoding with `abi.encodeCall`. It checks whether the supplied values actually match the types expected by the called function and also avoids errors caused by typos.

Update: *Resolved at commit 247f572, pull request #54.*

Notes & Additional Information

N-01 Unnecessary Variables

There are a few unnecessary variables in the codebase:

- At line 165 of OmnichainProposalSender, the variable `execution` is not needed since it is only used immediately in the next line.
- At line 164 of OmnichainProposalSender, the state variable `_lastStoredPayloadNonce` is not needed. Instead, the `pid` can be used as the mapping key for failed proposals.

Consider refactoring these unnecessary variables to increase code efficiency.

Update: *Resolved at commit e425804, pull request #54. The Venus team stated:*

> __lastStoredPayloadNonce : I would keep it as it is to track the number of failed messages that are retried or need to be._

N-02 Lack of Indexed Event Parameters

Throughout the codebase, several events do not have their parameters indexed:

- The `SetMaxDailyReceiveLimit` event of `BaseOmnichainControllerDest.sol`
- The `SetMaxDailyLimit` event of `BaseOmnichainControllerSrc.sol`
- The `FunctionRegistryChanged` event of `OmnichainExecutorOwner.sol`
- The `ReceivePayloadFailed` event of `OmnichainGovernanceExecutor.sol`
- The `UpdatedValidChainId` event of `OmnichainProposalSender.sol`

Consider indexing event parameters to improve the ability of off-chain services to search for and filter specific events.

Update: *Resolved at commit 9a80eeb, pull request #54. The Venus team stated:*

> *Not needed for uint256*

N-03 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) in a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so.

In addition, if a contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for the maintainers of those libraries to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact. For instance:

- The `BaseOmnichainControllerDest` abstract contract.
- The `BaseOmnichainControllerSrc` contract.
- The `IGovernanceBravoDelegate` interface.
- The `ITimelock` interface.
- The `IOmnichainGovernanceExecutor` interface.
- The `OmnichainExecutorOwner` contract.
- The `OmnichainGovernanceExecutor` contract.
- The `OmnichainProposalSender` contract.

Consider adding a NatSpec comment containing a security contact above the contract definitions. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved at commit 2b3e72c, pull request #54.*

N-04 Using `int/uint` Instead of `int256/uint256`

In `ITimelock.sol`, there are multiple instances of `int/uint` being used as opposed to `int256/uint256`:

- At line 8 of `ITimelock.sol`
- At line 13 of `ITimelock.sol`
- At line 37 of `ITimelock.sol`
- At line 40 of `ITimelock.sol`
- At line 53 of `ITimelock.sol`

- At line 56 of `ITimelock.sol`
- At line 69 of `ITimelock.sol`
- At line 72 of `ITimelock.sol`

In favor of explicitness, consider replacing all instances of `int/uint` with `int256/uint256`.

Update: *Resolved at commit d90f9b3, pull request #54.*

N-05 Hash Operation Only Necessary When Call Fails

At line 192 of `OmnichainGovernanceExecutor`, `payload_` is hashed and then stored in the `failedMessages` mapping. However, this operation is only needed when the function fails to call `nonblockingLzReceive()`.

In order to save gas and avoid unnecessary computation, consider moving this process inside the `if` statement at line 201.

Update: *Resolved at commit feb679d, pull request #54.*

N-06 The File Name and the Contract Name Mismatch

The `IGovernananceBravoDelegate` file name does not match the `IGovernanceBravoDelegate` contract name.

Consider renaming the files to match the contract names to make the codebase easier to understand for developers and reviewers.

Update: *Resolved at commit 162829b, pull request #54.*

N-07 Unnecessary Internal Function

The `_getFunctionName` function in the `OmnichainExecutorOwner` contract is only used once at line 53, and it only performs one look-up from the `functionRegistry` storage variable. As such, there is little need to use a stand-alone internal function to retrieve this value.

Consider removing this internal function and instead directly checking `functionRegistry` at line 53.

Update: *Resolved at commit ff169d3, pull request #54.*

N-08 Unnecessary Variable Casts

Throughout the codebase, several variables are casted unnecessarily:

- In the `OmnichainExecutorOwner` contract, the `accessControlManager_` variable in the `initialize` function.
- In the `OmnichainExecutorOwner` contract, the `newOwner_` variable in the `transferBridgeOwnership` function.
- In the `OmnichainGovernanceExecutor` contract, the `proposal.proposalType` variable in the `_queue` function.

To improve the overall clarity, intent, and readability of the codebase, consider removing unnecessary casts.

Update: *Resolved at commit 25cf0a5, pull request #54.*

N-09 Immutables Not Using `UPPER_CASE` Format

Throughout the codebase, several immutable variables do not use the `UPPER_CASE` format:

- The `omnichainGovernanceExecutor` constant declared at line 22 of `OmnichainExecutorOwner.sol`
- The `guardian` constant declared at line 51 of `OmnichainGovernanceExecutor.sol`
- The `lzEndpoint` constant declared at line 54 of `OmnichainProposalSender.sol`

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating the words. For better readability, consider following this convention.

Update: *Resolved at commit 6410b21, pull request #54.*

N-10 Unused Enum

In `OmnichainProposalSender.sol`, the `ProposalType` enum is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused enums.

Update: *Resolved at commit 8959c82, pull request #54.*

N-11 Inconsistent Use of Named Returns

To improve the readability of a smart contract, use the same return style in all of its functions.

The `OmnichainProposalSender` contract has inconsistent usage of named returns in its functions.

Consider using named returns in all functions.

Update: *Resolved at commit c18e413, pull request #54.*

N-12 Unused Enum Types

Throughout the codebase, there are unused enum types:

- In the `ProposalType` enum, the `NORMAL` type
- In the `ProposalType` enum, the `FASTTRACK` type
- In the `ProposalType` enum, the `NORMAL` type
- In the `ProposalType` enum, the `FASTTRACK` type
- In the `ProposalType` enum, the `CRITICAL` type

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused enum types.

Update: *Resolved. The Venus team stated:*

> *For Sender contract it is fixed in N-10 and for executor it is used here https://github.com/VenusProtocol/governance-contracts/blob/7304a686cbba52cdd2636ed8f0dbaa82adb582f6/contracts/Cross-chain/OmnichainGovernanceExecutor.sol#L125*

Conclusion

The Venus protocol aims to extend its governance system to different chains. The current system on the primary chain is to remain untouched. However, the proposal voting mechanism has now been extended, allowing proposals to be sent across chains.

The protocol has decided to use LayerZero for cross-chain messaging. LayerZero is a well-documented and battle-tested cross-chain bridge that also provides example contracts for governance systems. In the event that the bridge is compromised, Venus mitigates proposal hijacking by adding command limits and timelocks. The Venus extension of the LayerZero governance contracts is thoughtful in design and follows the best practices for sending cross-chain messages.

The audit yielded some high-severity issues along with some medium and low-severity ones. In addition, recommendations aimed at improving the readability and clarity of the codebase have also been made. The Venus team was highly responsive throughout the audit period and answered any questions we had regarding the protocol.