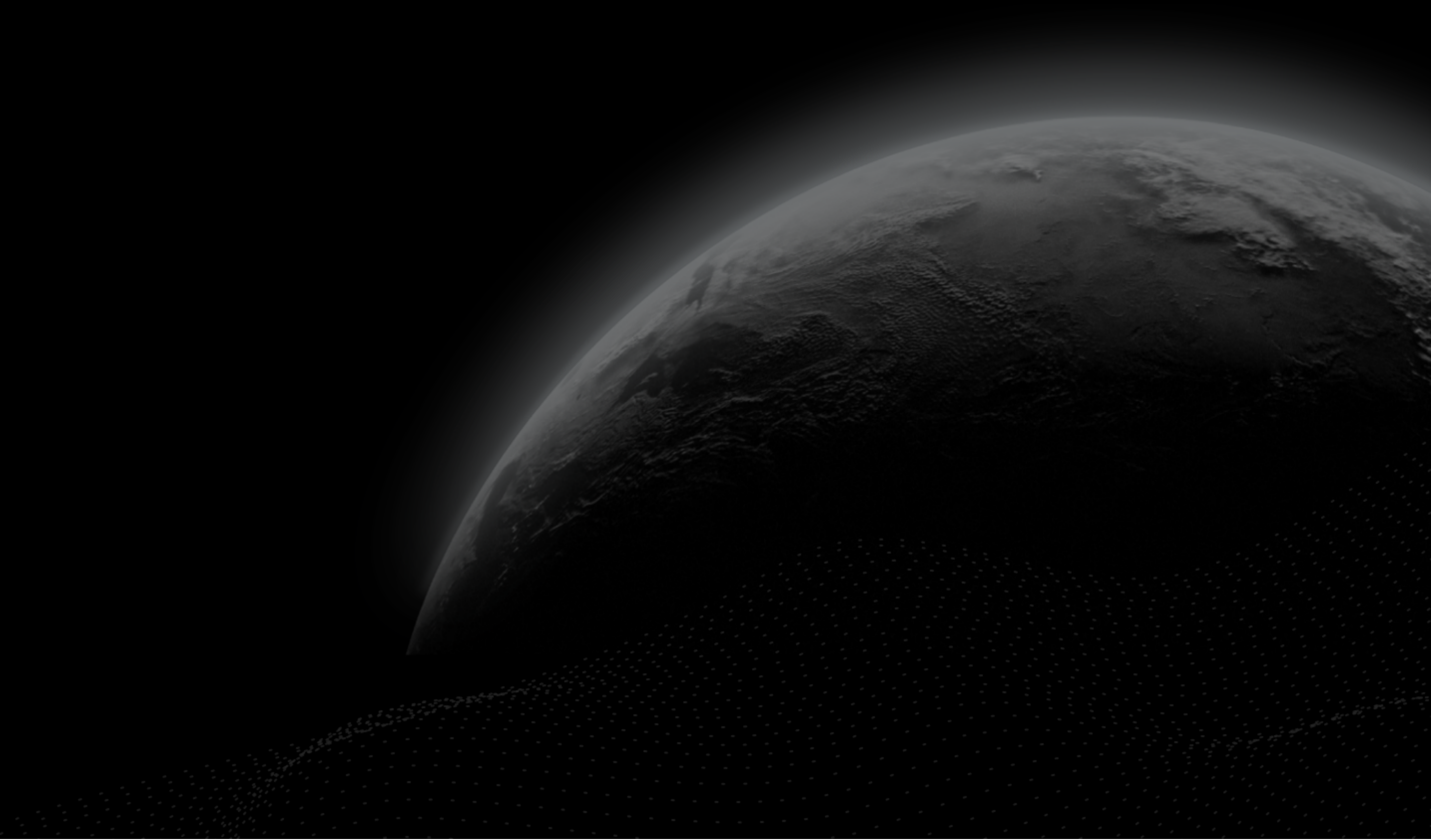
CERTIK

Security Assessment

Venus - Multichain Governance

CertiK Assessed on Feb 26th, 2024



CertiK Assessed on Feb 26th, 2024

Venus - Multichain Governance

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Binance Smart Chain (BSC)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 02/26/2024	N/A

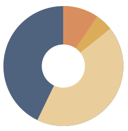
CODEBASE
https://github.com/VenusProtocol/governance-contracts
View All in Codebase Page

COMMITS
Base-PR: 13673edfe3578336f13b05b2954bd08ce09189b0
Base-Timelock: efaf4260a5772a88f4a6368084807c1a7dbecebb
Update1: dac3ce6663d36e0ebe54416c0fe24199763437e2
View All in Codebase Page

Highlighted Centralization Risks

ⓘ Contract upgradeability

Vulnerability Summary



21	17	0	1	3	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.
🟧 2	Major	2 Acknowledged	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.
🟨 1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 9	Minor	8 Resolved, 1 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.

■ 9	Informational	8 Resolved, 1 Acknowledged	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 | VENUS - MULTICHAIN GOVERNANCE

CODEBASE | VENUS - MULTICHAIN GOVERNANCE

Repository

https://github.com/VenusProtocol/governance-contracts

Commit

Base-PR: 13673edfe3578336f13b05b2954bd08ce09189b0

Base-Timelock: efaf4260a5772a88f4a6368084807c1a7dbecebb

Update1: dac3ce6663d36e0ebe54416c0fe24199763437e2

Update2: 0e2348ed192730cd74091d997f867803d56cde9b

AUDIT SCOPE | VENUS - MULTICHAIN GOVERNANCE

7 files audited ● 5 files with Acknowledged findings ● 1 file with Resolved findings ● 1 file without findings

ID	Repo	File	SHA256 Checksum
● BOC	VenusProtocol/governance-contracts	Cross-chain/BaseOmnichainControllerDest.sol	d18877f2185b1bc4263bd5c68f6adccecda3e4dd50fbde7130e4fbebb7d7840a7
● BOS	VenusProtocol/governance-contracts	Cross-chain/BaseOmnichainControllerSrc.sol	e8c6cd90aa852b51fd57c0261cf19cb2a4071bcd01edbb2b288b54040d77e188
● OEO	VenusProtocol/governance-contracts	Cross-chain/OmnichainExecutorOwner.sol	6db7d758a90c55fdd849746ba16636baee81d11f84300c38e41185048c599be8
● OGE	VenusProtocol/governance-contracts	Cross-chain/OmnichainGovernanceExecutor.sol	574111d92259d6540d092e505879ae4997e1e08692f0d27ed14ef9c8309fa2dd
● OPS	VenusProtocol/governance-contracts	Cross-chain/OmnichainProposalSender.sol	04506c485d103c7ac6b5c4d6b7d00ebc6ad1964adb95e7100269b5f2ef6a5528
● ITC	VenusProtocol/governance-contracts	Cross-chain/interfaces/ITimelock.sol	289a51f24cdfb6fc77c6a775fb6afcb2460c8765ccee16b7107c4e294d44f23a
● TVG	VenusProtocol/governance-contracts	TimelockV8.sol	412554faf744b827b2d30ca43688ac5348b74f8df2ca3e78c4a7032cffe3f810

APPROACH & METHODS | VENUS - MULTICHAIN GOVERNANCE

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

SUMMARY | VENUS - MULTICHAIN GOVERNANCE

This audit concerns the changes made in files outlined in:

- PR-21 until commit 13673edfe3578336f13b05b2954bd08ce09189b0.

In addition the audit scope included the file TimelockV8 at commit efaf4260a5772a88f4a6368084807c1a7dbecebb.

Note that any centralization risks present in the existing codebase before these PRs were not considered in this audit and only those added in these PRs are addressed in the audit. We recommend all users to carefully review the centralization risks, much of which can be found in our previous audits which can be found here: https://skynet.certik.com/projects/venus.

DEPENDENCIES | VENUS - MULTICHAIN GOVERNANCE

Third Party Dependencies

The protocol is serving as the underlying entity to interact with third party protocols. The third parties that the contracts interact with are:

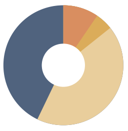
- Layer Zero endpoint relayers
- Third Party Contracts called during execution of proposals

The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. Moreover, updates to the state of a project contract that are dependent on the read of the state of external third party contracts may make the project vulnerable to read-only reentrancy. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

Recommendations

We recommend constantly monitoring the third parties involved to mitigate any side effects that may occur when unexpected changes are introduced, as well as vetting any third party contracts used to ensure no external calls can be made before updates to its state.

FINDINGS | VENUS - MULTICHAIN GOVERNANCE



21	0	2	1	9	9
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Venus - Multichain Governance. Through this audit, we have uncovered 21 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
OEO-02	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
VPB-02	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
OGE-08	Potential Reentrancy Attack	Logical Issue	Medium	Resolved
CRO-06	Missing Input Validation	Volatile Code	Minor	Partially Resolved
OEO-03	Unprotected Initializer	Coding Issue	Minor	Resolved
OPS-01	Eligibility May Be Expired At Time `retryExecute()` Is Called	Logical Issue	Minor	Resolved
OPS-02	Unnecessary `Payable` Casting May Lead To Locked Native Tokens	Logical Issue	Minor	Resolved
OPS-03	Issues With `ValidChainIds`	Logical Issue	Minor	Resolved
OPS-04	Case Not Handled By `fallbackWithdraw()`	Volatile Code	Minor	Resolved
OPS-07	Potential Issues With `msg.sender` Being Refunded In `retryExecute()`	Design Issue, Logical Issue	Minor	Resolved
OPS-08	Inaccurate Payload	Inconsistency	Minor	Resolved

ID	Title	Category	Severity	Status
VPB-06	Missing Proposal Status Checks And Updates	Volatile Code	Minor	● Resolved
BOC-01	Specific Imports Not Consistently Used	Inconsistency	Informational	● Resolved
CRO-04	Typos And Inconsistencies	Inconsistency	Informational	● Resolved
CRO-05	Event Should Include Chain Id	Coding Issue	Informational	● Resolved
CRO-07	Check-Effect-Interaction Pattern Violated	Volatile Code	Informational	● Resolved
OEO-04	Interface Defined In Same File It Is Used	Coding Issue	Informational	● Resolved
OGE-05	`whenNotPaused` Could Revert Sooner	Logical Issue	Informational	● Acknowledged
TVG-01	`delete` Keyword Can Be Used	Inconsistency	Informational	● Resolved
VPB-04	Previous State Variable Value Can Be Included In Event	Coding Issue	Informational	● Resolved
VPB-07	Missing Or Incomplete NatSpec	Inconsistency	Informational	● Resolved

OEO-02 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● **Major**	**Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 45~48**	● **Acknowledged**

Description

The contract `OmnichainExecutorOwner` is upgradeable; the corresponding `admin` role in each respective proxy has the authority to update the implementation contract behind each contract.

Any compromise to the `admin` account in each proxy may allow a hacker to take advantage of this authority and change the implementation contract the proxy points to, and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a 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 migrate to a new implementation contract.

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 combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- 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 deployed time-lock address.
- 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 combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

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

- Provide the deployed time-lock address.
- 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:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Alleviation

`[Venus, 02/05/2024]` : "The admin of the contracts on the new chains will a ProxyAdmin contract.

The owner of this ProxyAdmin contract will be a TimelockV8 contract, with the time configuration of the Normal Timelock used to execute the normal Venus Improvement Proposals (VIP) on BNB chain. For normal VIPs, the time configuration is: 24 hours voting + 48 hours delay before the execution.

So, these contracts will be upgraded only via a Normal VIP, involving the Venus Community/Governance in the process."

`[CertiK, 02/12/2023]` : In order to mitigate the finding completely, please provide the relevant information corresponding the new networks when they are available.

VPB-02 | CENTRALIZATION RELATED RISKS

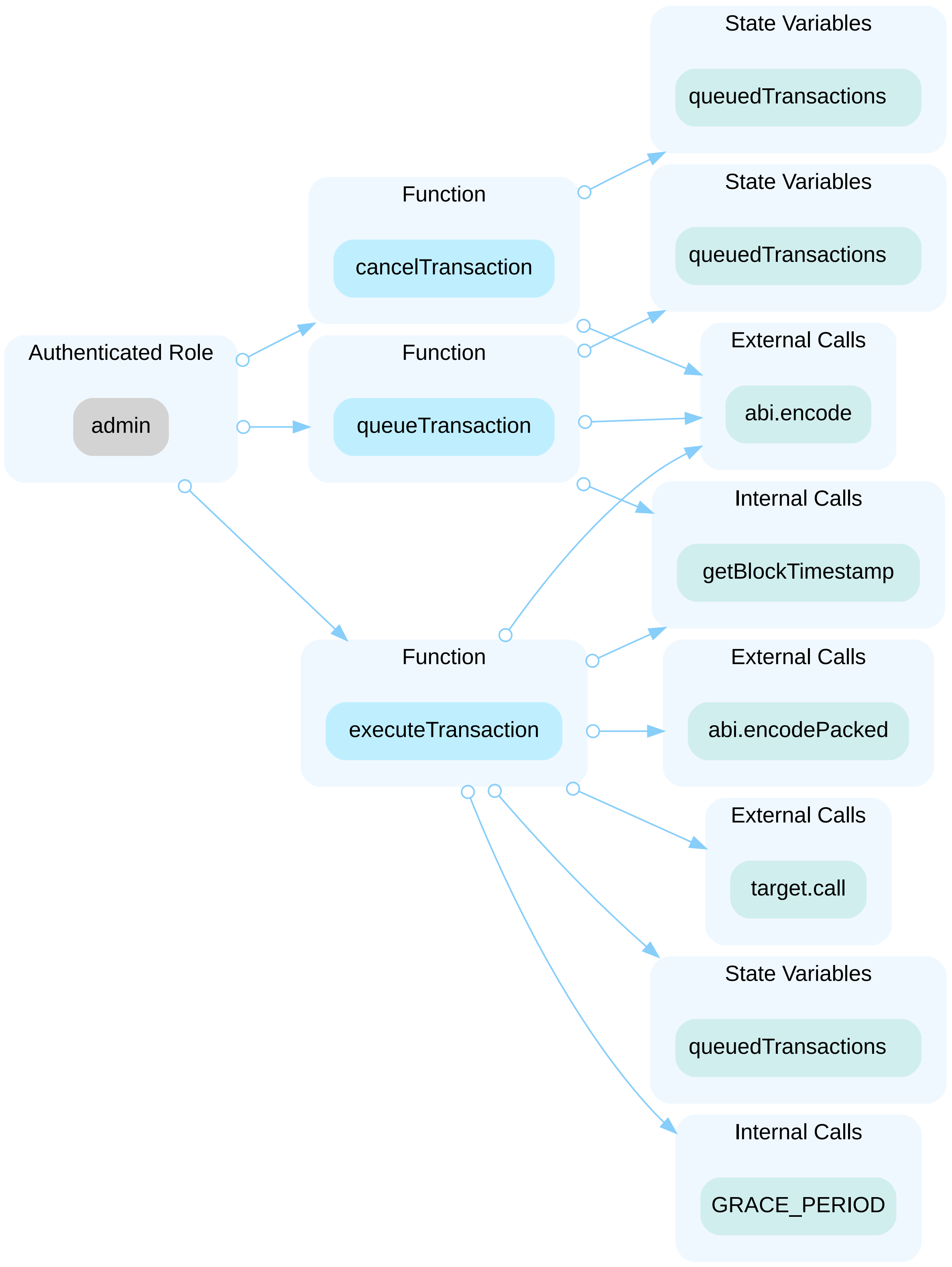
Category	Severity	Location	Status
Centralization	● **Major**	**Cross-chain/BaseOmnichainControllerDest.sol (Base-PR): 49~50, 58~59, 66~67; Cross-chain/BaseOmnichainControllerSrc.sol (Base-PR): 64~65, 74~75, 83~84, 94~95; Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 53~54, 68~69, 90~91; Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 121, 168, 172; Cross-chain/OmnichainProposalSender.sol (Base-PR): 117~118, 203~204, 239~240, 255~256, 265~266, 277~278; TimelockV8.sol (Base-Timelock): 112, 141, 169, 192**	● **Acknowledged**

Description

TimelockV8

In the contract `TimelockV8` the role `admin` has authority over the functions shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and do the following:

- Queue a malicious transaction.
- Cancel a transaction causing it to have to be queued again.
- Execute a transaction when it was not intended to be or execute a malicious transaction they had previously queued.



In addition to this, they also have control over the following functions which can only be called by queueing a transaction that will call them and then executing that transaction:

- `setDelay()`
- `setPendingAdmin()`

Any compromise to the `admin` account may allow a hacker to queue and then execute a transaction that will change the delay between the MINIMUM_DELAY and MAXIMUM_DELAY. It will also allow them to set the pending admin to an account they control, which can then call `acceptAdmin()` so that the team cannot recover the role from the hacker.

BaseOmnichainControllerSrc

In the contract `BaseOmnichainControllerSrc` the `DEFAULT_ADMIN_ROLE` of the `AccessControlManager` can grant addresses the privilege to call the following functions:

- `setMaxDailyLimit()`
- `pause()`
- `unpause()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or accounts granted this privilege may allow the hacker to take advantage of this authority and do the following:

- Update the max daily receive limit to 0 in order to prevent sending any proposals, or set it to a high value to allow more proposals to be sent than intended.
- Pause or unpause the contract during a crucial period.

In the contract `BaseOmnichainControllerSrc` , the role `_owner` has authority over the following functions:

- `setAccessControlManager()`

Any compromise to the `_owner` account may allow a hacker to take advantage of this authority and update the `accessControlManager` address to a malicious contract and gain control over setting the privilege of the functions above.

BaseOmnichainControllerDest

In the contract BaseOmnichainControllerDest, the role `_owner` has authority over the following functions:

- `setMaxDailyReceiveLimit()` ;
- `pause()` ;
- `unpause()` ;

Any compromise to the `_owner` account may allow a hacker to take advantage of this authority and

- Update the max daily receive limit to 0 in order to prevent receiving any proposals, or set it to a high value to allow more proposals to be accepted than intended.
- Pause or unpause the contract during a crucial period.

OmnichainExecutorOwner

In the contract `OmnichainExecutorOwner` the role `_owner` has authority over the following functions:

- `upsertSignature()`

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and make a function signature active within the `functionRegistry` mapping in order to give their account access to this Omnichain Governance logic through the privilege outlined below.

In the contract `OmnichainExecutorOwner` the role `DEFAULT_ADMIN_ROLE` of the `AccessControlManager` can grant addresses the privilege to call the following functions:

- Functions within the Omnichain Governance via the `fallback()` function in `OmnichainExecutorOwner`
- `transferBridgeOwnership()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or accounts granted this privilege may allow the hacker to take advantage of this authority and do the following:

- Call privileged functions that the `OmnichainExecutorOwner` controls within the Omnichain Governance.
- Give an account they control the owner privilege of the bridge in order to access other functions so they can take steps to perform the above.

OmnichainProposalSender

In the contract `OmnichainProposalSender` the role `_owner` has authority over the following functions:

- `fallbackWithdraw()`

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and clear a message that was intended to be retried and recover the `msg.value` for themselves.

In the contract `OmnichainProposalSender` the role `DEFAULT_ADMIN_ROLE` of the `AccessControlManager` can grant addresses the privilege to call the following functions:

- `execute()`
- `setTrustedRemoteAddress()`
- `setConfig()`
- `setSendVersion()`
- `updateValidChainId()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or accounts granted this privilege may allow the hacker to take advantage of this authority and do the following:

- Submit a malicious proposal and send it to the remote chain so that it can be executed there.
- Change the remote message receiver address to cause a denial of service or to execute malicious functionality.
- Change the configuration of the LayerZero messaging library to cause a denial of service.
- Update the list of valid chain Ids to cause a denial of service or support a chain that is not intended to be supported.

OmnichainGovernanceExecutor

In the contract `OmnichainGovernanceExecutor` the role `_owner` has authority over the following functions:

- `addTimelocks()`

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the timelocks so that proposals can either be executed much sooner or much later than anticipated.

In the contract `OmnichainGovernanceExecutor` the role `GUARDIAN` has authority over the following functions:

- `cancel()`

Any compromise to the `GUARDIAN` account may allow the hacker to take advantage of this authority and cancel proposals before they can be executed to cause a denial of service.

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

`[Venus, 02/05/2024]` : The admin of the TimelockV8 contracts will be the instance of the GovernorBravoDelegator contract deployed to the destination chain.

The owner of the BaseOmnichainControllerSrc (and OmnichainProposalSender) on BNB chain will be 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396, that is the Timelock contract used to execute the normal Venus Improvement Proposals (VIP).

The owner of the OmnichainExecutorOwner contract, on the new networks, will be the TimelockV8 contract deployed to that network equivalent to the Normal one on BNB chain.

The owner of the BaseOmnichainControllerDest (and OmnichainGovernanceExecutor) on the new chains will be an instance of the OmnichainExecutorOwner contract.

The GUARDIAN account will be a multisig wallet, with a minimum 3/6 configuration.

On BNB chain, we'll use the AccessControlManager (ACM) deployed at 0x4788629abc6cfca10f9f969efdeaa1cf70c23555. In this ACM, only 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396 (Normal Timelock) has the DEFAULT_ADMIN_ROLE.

And this contract is a Timelock contract used during the Venus Improvement Proposals. We'll use a similar setup on the new networks.

On BNB chain, we'll grant [a] (Normal), [b] (Fast-track) and [c] (Critical) timelocks to execute the following functions:

- BaseOmnichainControllerSrc.setMaxDailyLimit()
- BaseOmnichainControllerSrc.pause()
- BaseOmnichainControllerSrc.unpause()
- OmnichainProposalSender.execute()
- OmnichainProposalSender.setConfig()
- OmnichainProposalSender.setSendVersion()
- OmnichainProposalSender.updateValidChainId()

Only the [a] Normal timelock will be granted to execute this:

- OmnichainProposalSender.setTrustedRemoteAddress()

On the new chains, we'll grant these permissions:

- OmnichainExecutorOwner.transferBridgeOwnership(), to the Normal timelock
- OmnichainGovernanceExecutor.setSrcChainId, to the Normal timelock
- OmnichainGovernanceExecutor.addTimelocks, to the Normal timelock

The current config for the three Timelock contracts on BNB chain are:

- normal: 24 hours voting + 48 hours delay
- fast-track: 24 hours voting + 6 hours delay
- critical: 6 hours voting + 1 hour delay

For the new networks we plan a similar setup (three timelocks with these time constraints)

[a] 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396

[b] 0x555ba73dB1b006F3f2C7dB7126d6e4343aDBce02

[c] 0x213c446ec11e45b15a6E29C1C1b402B8897f606d

`[CertiK, 02/12/2023]` : The client has provided all steps towards mitigation on the BSC chain. In order to mitigate the finding completely, please provide the relevant information corresponding the new networks when they are available.

OGE-08 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Logical Issue	● Medium	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 141~160	● Resolved

Description

The function `execute()` does not follow check-effect-interact and the transactions executed may make external calls allowing for potential reentrancy. In particular, if there is more than one proposal queued, then this can allow a second proposal to be executed in the middle of the execution of another proposal. Considering the proposal executed in the middle of the other proposal can execute privileged actions that may not have been accounted for, this could cause potential harm to the protocol.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern and adding the `nonReentrant` modifier.

Alleviation

`[CertiK, 02/12/2024]`: The client made changes resolving this finding in commit dd7ece9a31546f89d468d23af77b46d676989896.

CRO-06 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 112~113, 131~132, 216~222; Cross-chain/OmnichainProposalSender.sol (Base-PR): 117~118	● Partially Resolved

Description

OmnichainGovernanceExecutor.sol

- The constructor is missing a check that `guardian_` is a nonzero address.
- The function `addTimelocks()` is missing a check to ensure that each of the entries of `proposalTimelocks` is distinct by the end of all updates.
- The function `addTimelocks()` is missing a check to prevent updating to a new timelock while transactions have been queued but not executed or canceled
- The function `_nonblockingLzReceive()` is missing a check that all arrays are the same length.
- Function `_nonblockingLzReceive()` is missing a check that the `proposalType` decoded in the sent payload is a value between 0 and 2, inclusive. Anything larger will correspond to a `proposalType` that is not implemented.

OmnichainProposalSender.sol

- Function `execute()` is missing a check that all arrays are the same length.
- Function `execute()` is missing a check that the `proposalType` decoded in the sent payload is a value between 0 and 2, inclusive. Anything larger will correspond to a `proposalType` that is not implemented.

Recommendation

We recommend adding in the missed checks outlined above.

Alleviation

`[CertiK, 02/12/2024]`: The client made changes partially resolving the finding in commits

- 01f6b9902e0a312b18891c2d06077f4b56345922
- 750908bed01dbf37100b9d9351a61003ac171058
- 2c94bd93f3559aa50688abcad257dac9d5f9c645

1. The client opted not to add a check to `addTimelocks()` ensuring that each of the entries of `proposalTimelocks` is distinct by the end of all updates;
2. The client opted not to add a check to `addTimelocks()` preventing the update to a new timelock while transactions have been queued but not yet executed or canceled;
3. The client opted not to add a check to `execute()` ensuring that the `proposalType` is between 0 and 2 inclusive (this check has been added to `_nonblockingLzReceive()` however).

`[Venus, 02/15/2024]` : "Regarding the extra checks on addTimelocks(), we prefer to keep it as it is. It's a privilege function, that only will be executable from Governance, so there should be enough time to review and simulate it."

OEO-03 | UNPROTECTED INITIALIZER

Category	Severity	Location	Status
Coding Issue	● Minor	Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 36~37	● Resolved

Description

The logic contract `OmnichainExecutorOwner` does not protect its initializer function. An attacker can call the initializer and assume ownership of the logic contract, whereby she can perform privileged operations that trick unsuspecting users into believing that she is the owner of the upgradeable contract.

Recommendation

We recommend calling `_disableInitializers` in the constructor or giving the constructor the `initializer` modifier to prevent the intializer from being called on the logic contract.

Reference: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Alleviation

`[CertiK, 02/12/2024]` : The client made changes resolving this finding in commit cba2e74107959193d92232874314ab79ed611230.

OPS-01 | ELIGIBILITY MAY BE EXPIRED AT TIME `retryExecute()` IS CALLED

Category	Severity	Location	Status
Logical Issue	● Minor	Cross-chain/OmnichainProposalSender.sol (Base-PR): 164~165	● Resolved

Description

Function `_isEligibleToSend()` is used as a check in function `execute()`. However, if the proposal is not properly executed, it gets stored, and the execution can be retried at a later time. The checks made in `_isEligibleToSend()` are based on a 24-hour window. If the second attempts are made outside the 24 hour window in which the execution was initially tried, then it is possible that the number of proposals exceeds the limit during the retry.

Additionally, note that when proposal messages are retried, it may result in two proposals sent during the same `block.timestamp`, a scenario that is supposed to be prevented by function `_isEligibleToSend()`.

Recommendation

We recommend providing a plan for prompt execution or canceling of failed messages, to ensure they are handled within the same 24 hour window. Additionally, please specify whether it is acceptable that a retried failed message would be executed potentially in the same `block.timestamp` as another proposal.

Alleviation

`[Venus, 02/01/2024]`: "We are now re-validating the message in the retryExecute function, so _isEligibleToSend() is executed again."

`[CertiK. 02/12/2024]`: The client made changes resolving this finding in commit dac3ce6663d36e0ebe54416c0fe24199763437e2.

OPS-02 | UNNECESSARY `Payable` CASTING MAY LEAD TO LOCKED NATIVE TOKENS

Category	Severity	Location	Status
Logical Issue	● Minor	Cross-chain/OmnichainProposalSender.sol (Base-PR): 210~211	● Resolved

Description

Function `fallbackWithdraw()` is a privileged function used to remove failed proposals that cannot be executed. The function includes a `payable` casting, even though it must be used to remove proposals and remove the corresponding amount of funds from the contract. If the `_owner` includes a `msg.value` with the call, then this amount could become stuck in the contract, since it does not correspond to a proposal.

Recommendation

We recommend removing the `payable` casting from the function `fallbackWithdraw()` .

Alleviation

`[CertiK, 02/12/2024]` : The client made changes resolving this finding in commit 0bd5d19415cc53baff842faec05fe74608d5fdef.

OPS-03 | ISSUES WITH `ValidChainIds`

Category	Severity	Location	Status
Logical Issue	● Minor	Cross-chain/OmnichainProposalSender.sol (Base-PR): 124, 239	● Resolved

Description

The mapping `validChainIds` is used to determine if an input remote chain id is valid. However, the address of the contract on the remote chain to receive messages sent by this contract should only be set if the remote chain is valid.

Only `setTrustedRemoteAddress()` is implemented and there is no method such as `setTrustedRemote()` that can set set it to the default empty path. Thus once set `trustedRemoteLookup[remoteChainId_].length` will always be nonzero. `validChainIds` can then invalidate a remote chain in the event that the remote address was set and it needs to be removed as a valid chain.

Alternatively, `validChainIds` can be removed and instead the `setTrustedRemote()` method (see here) can be implemented so that in the event that a remote chain should be invalidated `trustedRemoteLookup[remoteChainId_]` can be set to the empty path and the check

```
require(trustedRemote.length != 0, "OmnichainProposalSender: destination chain is
not a trusted source");
```

would always ensure that the remote chain is valid.

Function `retryExecute()` should include a relevant check based on the above to ensure the input `remoteChainId_` is still allowed at the time of the second attempt.

Recommendation

We recommend considering the removal of `validChainIds` and instead adding a method to set a trusted remote to the empty path. In addition, we recommend ensuring that `retryExecute()` has sufficient checks to ensure that the input `remoteChainId_` is allowed.

Alleviation

`[CertiK, 02/12/2024]` : The client made changes resolving this finding in commit 42afbc93c676f5794295227016c071e69899f7c5.

OPS-04 | CASE NOT HANDLED BY `fallbackWithdraw()`

Category	Severity	Location	Status
Volatile Code	● Minor	Cross-chain/OmnichainProposalSender.sol (Base-PR): 212~213	● Resolved

Description

Function `execute()` allows a message to be executed with a `msg.value` of 0, which may lead to a failed message. Such a message may need to be removed, but cannot be removed via function `fallbackWithdraw()` , due to the requirement that `originalValue_` input is nonzero. As a result, a failed message of this kind may become stuck in `OmnichainProposalSender` .

Recommendation

We recommend including an additional `cancel()` function which allows a privileged account to clear failed messages that included an `originalValue_` of 0, ie, failed messages that may not include a withdraw of funds.

Alleviation

`[CertiK, 02/12/2024]` : The client updated `execute()` so that it must have a `msg.value` greater than zero in commit ac564b369537095d6be36c7b1fecebf91d51ccc6.

OPS-07 | POTENTIAL ISSUES WITH `msg.sender` BEING REFUNDED IN `retryExecute()`

Category	Severity	Location	Status
Design Issue, Logical Issue	● Minor	Cross-chain/OmnichainProposalSender.sol (Base-PR): 142, 184~185	● Resolved

Description

Function `retryExecute()` can be called by anyone, because there is a check ensuring the payload entered was one previously attempted at execution, but which failed.

In `retryExecute()`, the refund is directed to go to the `msg.sender` currently making the call, even though the account that initially providing the `msg.value` was the account that called function `execute()`. The account attempting `retryExecute()` may not be the original account that called `execute()`, since this function is free to be called by anyone.

It is possible that, in the original call to `execute()`, a large `msg.value` was included, with the idea being any difference would be refunded; it is possible that the execution fails for some other reason besides insufficient fees, however. In such a case, someone may watch for such a failure and attempt to call `retryExecute()` with the opportunity to seize the refund, which may be significant.

Additionally, a low-level call is made to the refund recipient when sending the refund, and this is done before all logic has been completed by the endpoint. This may pose a system-reentrancy risk.

Recommendation

We recommend taking the above into consideration in deciding whether to make an unprotected `msg.sender` the refund recipient.

Alleviation

`[CertiK, 02/12/2024]`: The client made changes resolving this issue in commit 622ac5e11998cc21783a069a647adf31a45b7f98.

OPS-08 | INACCURATE PAYLOAD

Category	Severity	Location	Status
Inconsistency	● Minor	Cross-chain/OmnichainProposalSender.sol (Base-PR): 95~96, 159~160, 176~177, 196~197, 219~220	● Resolved

Description

The payload that is sent or stored at `storedExecutionHashes[_pId]` in the `execute()` function includes a second layer of encoding the pId with the initial `payload_`. The parameter comment in functions `retryExecute()` and `fallbackWithdraw()` here makes it seem as if the original `payload_` input in function `execute()` should be used again; this will result in a revert. The comments could either be rewritten to accurately specify the extra layer of encoding, or else this encoding step could be done within the logic to ensure the check will come out correctly when comparing the hashes.

The parameter `payload_` in the comments above `estimateFees()`, `retryExecute()` and `fallbackWithdraw()` are also missing the inclusion of the `proposalType` at the end of the encoding.

Recommendation

We recommend making one of the two adjustments outlined above.

Alleviation

`[CertiK, 02/12/2024]`: The client made changes resolving the finding in commit [c0e7529cdede8208a18a8042063b79e8da3929f3](https://github.com/VenusProtocol/governance-contracts/commit/c0e7529cdede8208a18a8042063b79e8da3929f3).

VPB-06 | MISSING PROPOSAL STATUS CHECKS AND UPDATES

Category	Severity	Location	Status
Volatile Code	● Minor	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 141~142, 168~169, 237~238, 244~245; TimelockV8.sol (Base-Timelock): 141~142, 178~179	● Resolved

Description

While it is understood that the in-scope `TimelockV8` contract and `OmnichainProposalSender` work together with the `OmnichainGovernanceExecutor` contract to ensure the proper intended behavior in proposal queuing, execution, and cancellation, the following checks should be added to the `OmnichainGovernanceExecutor` contract in order to ensure correct functioning in the event the timelock contracts are updated through `addTimelocks()`.

For instance, if the timelocks are ever updated, in the current `OmnichainGovernanceExecutor` contract, the fact that previously executed `proposals` remained set to `true` in `queued` may potentially allow for the possibility of replaying the execution of a proposal, since there are no checks within the `OmnichainGovernanceExecutor` contract to ensure the proposal has not already been executed or cancelled.

Additionally, currently there should be only one source chain providing proposals, so it is assumed that all incoming `pId` values have not previously been set. However, the `OmnichainGovernanceExecutor` contract has a mapping to keep track of limits for multiple source chains (`chainIdToMaxDailyReceiveLimit`). In the event that multiple source chains send proposals, the `pId` method may no longer be unique.

Checks are suggested below in order to ensure the `OmnichainGovernanceExecutor` and `TimelockV8` contracts can function properly, independent of one another.

OmnichainGovernanceExecutor.sol

- Function `execute()` is missing a check ensuring that `proposal.executed` is previously false before updating it to true.
- Function `execute()` is missing a check ensuring that `proposal.cancelled` is false before executing it.
- Function `execute()` should update `queued[proposalId_]` to false as part of the logic.
- Function `cancel()` is missing a check ensuring that `proposal.cancelled` is false before executing it.
- Function `cancel()` should update `queued[proposalId_]` to false as part of the logic.
- Function `_nonblockingLzReceive()` is missing a check ensuring that `proposals[pId]` is not already set before updating it.
- Function `_queue()` is missing a check ensuring that `queued[proposalId_]` is false before setting it to true.

TimelockV8.sol

- Function `queueTransaction()` is missing a check ensuring that `queuedTransactions[txHash]` is false before updating it to true.
- Function `cancelTransaction()` is missing a check ensuring that `queuedTransactions[txHash]` is true before updating it to false.

Recommendation

We recommend adding the checks outlined above.

Alleviation

`[CertiK, 02/12/2024]` : The client made changes partially resolving the finding in commits

- e4d298e6e253d71c1771ee507530fad9b57e8549
- c9915b6b24919793cc15badba1c5d408f5bcc315

The client opted not to add the missing checks outlined for `TimeklockV8.sol` .

Note that with the new `state()` checking pattern added in contract `OmnichainGovernanceExecutor` , proposals that have `queued[proposalId_]` set as false will now default to a return of `ProposalState.Canceled` . We recommend including an extra enum state to represent proposals that have not yet been queued.

`[Venus, 02/15/2024]` : "**Required checks on the TimelockV8:**

- d4108b44a3e9be70890f7e8148ae15a8db08e762
- 3c8b8d5a2ea90213f3c84a063b5c4531adcfa9fe

state() reverting if the proposal doesn't exist: we initially considered the option of adding a new state, but we finally decided to remove it and revert the `state()` call if the proposal doesn't exist (or it's in an unknown state, that should not be possible).

- 22f92af256cb9b919709bfc29764e00bfde4640c
- cdc41b666811019ac37360ae8666c2e429c93333"

BOC-01 | SPECIFIC IMPORTS NOT CONSISTENTLY USED

Category	Severity	Location	Status
Inconsistency	● Informational	Cross-chain/BaseOmnichainControllerDest.sol (Base-PR): 5	● Resolved

Description

Many of the added files use specific imports, however, some import the entire file.

Recommendation

We recommend using specific imports to clarify what is used and remain consistent.

Alleviation

`[CertiK, 02/08/2024]` : The client made changes resolving the finding in commit 08d5fa944ac465c5d8a9c364b34bb0005932fc0b.

CRO-04 | TYPOS AND INCONSISTENCIES

Category	Severity	Location	Status
Inconsistency	● Informational	Cross-chain/BaseOmnichainControllerDest.sol (Base-PR): 44~45; Cross-chain/BaseOmnichainControllerSrc.sol (Base-PR): 14, 60~61, 123~124; Cross-chain/interfaces/ITimelock.sol (Base-PR): 36~37, 53~54, 69~70	● Resolved

Description

BaseOmnichainControllerSrc.sol

- The comments above the contract `BaseOmnichainControllerSrc` have an additional "*" before the security-contact.
- In the comments above function `setMaxDailyLimit()` , parameter `limit_` is not representing an amount in USD.
- In the comments above function `setMaxDailyLimit()` , there is stray notation ")." included at the end of line `@param limit_` .
- The comment in function `_isEligibleToSend()` states multiple commands cannot be sent for a `dstChainId_` within a proposal, but this is allowed; it is that multiple proposals cannot be sent within the same `block.timestamp` .

BaseOmnichainControllerDest.sol

- In the comments above function `setMaxDailyReceiveLimit()` , parameter `limit_` is not representing an amount in USD.
- in the comments above function `setMaxDailyReceiveLimit()` the parameter `@param chainId_` says it is representing the destination chain ID, but it is actually representing source chain id.

ITimelock.sol

- In each of the comments cited above, the word "signature" is misspelled as "Ssignature."

Recommendation

We recommend correcting the typos and inconsistencies above.

Alleviation

`[CertiK, 02/12/2024]` : The client made changes resolving this finding in commit 3cc8aedf2a8417b4420e46dbedb2ce4b48ee077a.

CRO-05 | EVENT SHOULD INCLUDE CHAIN ID

Category	Severity	Location	Status
Coding Issue	● Informational	Cross-chain/BaseOmnichainControllerDest.sol (Base-PR): 36~37, 50~51; Cross-chain/BaseOmnichainControllerSrc.sol (Base-PR): 45~46, 66~67	● Resolved

Description

The event `SetMaxDailyLimit` should include the chain id to make it easier to track which chain's limit is being updated.

Recommendation

We recommend including the chain id in the emission of the event `SetMaxDailyLimit` .

Alleviation

`[CertiK, 02/08/2024]` : The client made changes resolving the finding in commit 55739852a484b01c894b18c609f05db6fd930cb8.

CRO-07 | CHECK-EFFECT-INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Volatile Code	● Informational	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 248~259; Cross-chain/OmnichainProposalSender.sol (Base-PR): 138~147, 180~188	● Resolved

Description

While the `nonreentrant` modifier and access restriction can prevent reentrancy, we recommend following check-effect-interaction wherever possible as added protection and best practice.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern wherever possible.

Alleviation

`[CertiK, 02/12/2024]`: The client made changes resolving the finding in commits

- f95e28d1ab3e4b34194013899c5391eb6b0c0aec;
- 4f3940a6b3fb2f639416948a946ac94b2dd45643.

OEO-04 | INTERFACE DEFINED IN SAME FILE IT IS USED

Category	Severity	Location	Status
Coding Issue	● Informational	Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 6~8	● Resolved

Description

The interface `IOmnichainGovernanceExecutor` in `OmnichainExecutorOwner` can be defined in a separate file so that it is easier for future projects to import them and avoid duplication.

Recommendation

We recommend defining the interface in a separate file and importing it.

Alleviation

`[CertiK, 02/08/2024]` : The team made changes resolving the finding in commit 64c4a642bee4f10c2f7d112b755aa5154d1b2077.

OGE-05 | `whenNotPaused` COULD REVERT SOONER

Category	Severity	Location	Status
Logical Issue	● Informational	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 193~194	● Acknowledged

Description

The function `_blockingLzReceive()` was overridden to include the `whenNotPaused` modifier. However, if the protocol is paused, then it will cause the transaction to revert without storing the failed message.

If it is intended to not store any messages when paused, then the `whenNotPaused` modifier can be moved to an overrided external facing `lzReceive()` function so that it reverts sooner.

Alternatively if messages should be stored when paused, then the `whenNotPaused` modifer can be moved to the `_nonblockingLzReceive()` function. However, if this is done a method should be added allowing failed messages to be canceled before unpausing in case malicious failed messages are stored and could be retried.

Also we recommend considering adding a method to cancel stored failed messages, because if a malicious proposal is stored as a failed message and then is retried successfully. Canceling each individual transaction could be much more intensive than canceling the failed messages proposal.

Recommendation

We recommend moving the `whenNotPaused` check to the external-facing `lzReceive()` function.

Alleviation

`[Venus, 02/01/2024]` : "It is intended to not store any messages when paused. While the impact of this change may be minimal, incorporating whenNotPaused with lzReceive() necessitates the overriding of an additional function. So, we prefer to keep it as it is.""

TVG-01 | `delete` KEYWORD CAN BE USED

Category	Severity	Location	Status
Inconsistency	● Informational	TimelockV8.sol (Base-Timelock): 179, 206	● Resolved

Description

In the functions `cancelTransaction()` and `executeTransaction()` the keyword `delete` can be used in place of setting `queuedTransactions[txHash]` to false.

Recommendation

We recommend using the `delete` keyword when setting variables to their default.

Alleviation

`[CertiK, 02/08/2024]` : The team made changes resolving the finding in commit 44368a767ee4262a02679113c5ba671bd3b674fd.

VPB-04 | PREVIOUS STATE VARIABLE VALUE CAN BE INCLUDED IN EVENT

Category	Severity	Location	Status
Coding Issue	● Informational	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 109~110; Cross-chain/OmnichainProposalSender.sol (Base-PR): 49~50; TimelockV8.sol (Base-Timelock): 14~15, 20~21	● Resolved

Description

In the events cited, it may be beneficial for on-chain reference to include the previous value of the updated state variable.

Recommendation

We recommend including the old state variable value in the event emissions cited.

Alleviation

`[CertiK, 02/08/2024]` : The team made changes resolving the finding in commit 786d296b4e0e90caeeabe3f6d8b4dd4dba580018

VPB-07 | MISSING OR INCOMPLETE NATSPEC

Category	Severity	Location	Status
Inconsistency	● Informational	Cross-chain/BaseOmnichainControllerDest.sol (Base-PR): 75; Cross-chain/BaseOmnichainControllerSrc.sol (Base-PR): 105, 132; Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 50~52; Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 188; Cross-chain/OmnichainProposalSender.sol (Base-PR): 108~116; Cross-chain/interfaces/ITimelock.sol (Base-PR): 65~72; TimelockV8.sol (Base-Timelock): 88, 97~107, 112, 124, 141, 169, 192, 225	● Resolved

Description

BaseOmnichainControllerDest

- There are no comments above the function `_isEligibleToReceive()`.

BaseOmnichainControllerSrc

- There are no comments above the function `_isEligibleToSend()`.
- There are no comments above the function `_ensureAllowed()`.

OmnichainProposalSender

- In function `execute()` there is a missing natspec comment specifying the access restriction of the function

OmnichainGovernanceExecutor

- There are no comments above the function `_blockingLzReceive()`.
- There are no comments above the function `_nonblockingLzReceive()`.
- There are no comments above the function `_queue()`.
- There are no comments above the function `_queueOrRevertInternal()`.

OmnichainExecutorOwner

- The comments above the `fallback()` function do not mention the access restriction, input, or return value.

ITimelock

- The comments above `executeTransaction()` do not include a description of the return value.

TimelockV8

- The comments above `setDelay()` do not mention the emitted event or access restriction.
- There are no comments above the functions `GRACE_PERIOD()`, MINIMUM_DELAY()`, `MAXIMUM_DELAY()`, or` getBlockTimestamp()`.
- The comments above `acceptAdmin()` do not mention the emitted event or access restriction.
- The comments above `setPendingAdmin()` do not mention the emitted event or access restriction.
- The comments above `queueTransaction()` do not mention the emitted event or access restriction.
- The comments above `cancelTransaction()` do not mention the emitted event or access restriction.
- The comments above `executeTransaction()` do not mention the emitted event, access restriction, or return value.
- The comments above `queueTransaction()` do not mention the emitted event or access restriction.

Recommendation

We recommend adding the missing or incomplete NatSpec comments mentioned above.

Alleviation

`[CertiK, 02/15/2024]`: The client made changes resolving this finding in commits

- 99e77698290405a4f82884dd93d3d6ee2a13cbbc
- 9275a73bc0337a78753284ca1398616c054d653b
- f310b733ab52629c248c0b1179afabed8d1f3d5e

OPTIMIZATIONS | VENUS - MULTICHAIN GOVERNANCE

ID	Title	Category	Severity	Status
CRO-03	Increment Optimizations	Gas Optimization, Code Optimization	Optimization	● Resolved
OEO-01	Hardcoded Boolean Can Be Used	Gas Optimization	Optimization	● Resolved
OGE-04	Unnecessary `uint8` Casting	Code Optimization	Optimization	● Resolved
OGE-06	`pId` Can Be Used Instead Of `newProposal.id`	Code Optimization, Gas Optimization	Optimization	● Resolved
OGE-09	Use Temporary Variable To Save Reading From Storage	Gas Optimization	Optimization	● Resolved
OGE-10	Mapping `proposalTimelocks` Can Be Optimized As A Fixed-Length Array	Code Optimization, Gas Optimization	Optimization	● Acknowledged
OPS-06	Unnecessary Check	Gas Optimization, Code Optimization	Optimization	● Resolved
TVG-02	Inefficient Memory Parameter	Inconsistency	Optimization	● Resolved

CRO-03 | INCREMENT OPTIMIZATIONS

Category	Severity	Location	Status
Gas Optimization, Code Optimization	● Optimization	Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 71~72; Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 126, 149~150, 175~176, 247~248	● Resolved

Description

In general, the counter in a for loop can be incremented or decremented in an unchecked block as it cannot overflow or underflow, saving gas as it will not perform a check for overflow or underflow.

Additionally, it saves a small amount of gas to increment an index in a `for` loop from the left instead of from the right side as it performs fewer operations.

OmnichainGovernanceExecutor.sol

- In function `addTimelocks()`, incrementing variable `i` can be incremented in an `unchecked` block.
- In function `execute()`, incrementing variable `i` can be incremented in an `unchecked` block.
- In function `cancel()`, incrementing variable `i` can be
  - incremented in an `unchecked` block;
  - incremented from the left;
  - implicitly initialized at 0, it does not need to be set to 0 explicitly;
- In function `_queue()`, incrementing variable `i` can be incremented in an `unchecked` block.

OmnichainExecutorOwner.sol

- In function `upsertSignature()`, incrementing variable `i` can be
  - incremented in an `unchecked` block;
  - incremented from the left;

Recommendation

We recommend adjusting the increment logic in order to optimize.

Alleviation

`[CertiK. 02/08/2024]` : The client made changes resolving the finding in commit 4826f6ef3d6034ff25f6cb7dcbf7cbb435e2d551.

OEO-01 | HARDCODED BOOLEAN CAN BE USED

Category	Severity	Location	Status
Gas Optimization	● Optimization	Cross-chain/OmnichainExecutorOwner.sol (Base-PR): 76, 79	● Resolved

Description

In the function `upsertSignature()` there is the following logic:

```
if (active_[i] && signature.length == 0) {
                functionRegistry[sigHash] = signatures_[i];
                emit FunctionRegistryChanged(signatures_[i], active_[i]);
            }
```

However, if this if-block is executed then necessarily `active_[i]` is true so that the emitted event can use `true` as opposed to reading the value.

Similarly in the following logic:

```
else if (!active_[i] && signature.length != 0) {
                delete functionRegistry[sigHash];
                emit FunctionRegistryChanged(signatures_[i], active_[i]);
            }
```

If this if-block is executed then necessarily `active_[i]` is false so that the emitted event can use `false` as opposed to reading the value.

Recommendation

We recommend hardcoding the booleans.

Alleviation

`[CertiK, 02/05/2024]` : The client made changes resolving the finding in commit 815581a8c4b4c0e3bdcb5c29782fa63bf5c33c5e.

OGE-04 | UNNECESSARY `uint8` CASTING

Category	Severity	Location	Status
Code Optimization	● Optimization	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 150	● Resolved

Description

The `proposal.proposalType` is already `uint8` type, it is not necessary to cast it explicitly.

Recommendation

We recommend removing the `uint8` casting.

Alleviation

`[CertiK, 02/05/2024]` : The client made changes resolving the finding in commit 9bb244f9515fdafd553a9a4ee72b046d6fd0e172.

OGE-06 | `pId` CAN BE USED INSTEAD OF `newProposal.id`

Category	Severity	Location	Status
Code Optimization, Gas Optimization	● Optimization	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 237	● Resolved

Description

In function `_nonblockingLzReceive()` , the `newProposal.id` value is used as a key in the `proposals` mapping. The `newProposal.id` is the `pId` ; the mapping can be set using the `pId` as the key instead.

Recommendation

We recommend using the `pId` as a key instead of the `newProposal.id` .

Alleviation

`[CertiK, 02/05/2024]` : The client made changes resolving the finding in commit 9c2470f612d114381c456106bad94150cfa28d3e.

OGE-09 | USE TEMPORARY VARIABLE TO SAVE READING FROM STORAGE

Category	Severity	Location	Status
Gas Optimization	● Optimization	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 149, 150, 155, 175, 176, 181, 247, 254	● Resolved

Description

OmnichainGovernanceExecutor

- In the function `execute()` , `proposalTimelocks[uint8(proposal.proposalType)]` , `proposal.targets.length` , and `proposal.eta` can be stored in a temporary variable to avoid being repeatedly read from storage.
- In the function `_queue` , `proposal.proposalType` can be stored in a temporary variable to avoid being repeatedly read from storage.
- In the function `cancel()` , `proposalTimelocks[proposal.proposalType]` , `proposal.targets.length` , and `proposal.eta` can be stored in a temporary variable to avoid being repeatedly read from storage.
- In the function `_queue()` , `proposal.targets.length` and `proposal.proposalType` can be stored in a temporary variable to avoid being repeatedly read from storage.

Recommendation

We recommend using a temporary variable to store the storage variable.

Alleviation

`[CertiK, 02/08/2024]` : The client made changes resolving the finding in commit a6a232622b53b4dad8e915eea149cdb9e798bc20.

OGE-10 | MAPPING `proposalTimelocks` CAN BE OPTIMIZED AS A FIXED-LENGTH ARRAY

Category	Severity	Location	Status
Code Optimization, Gas Optimization	● Optimization	Cross-chain/OmnichainGovernanceExecutor.sol (Base-PR): 67~68, 117~118, 119~120, 123~124, 126~127	● Acknowledged

Description

If the mapping `proposalTimelocks` is updated to a fixed-length array, deployment cost and the cost to call function `addTimelocks()` will increase slightly, but the calls to functions `execute()` and `cancel()` will decrease in gas cost.

Recommendation

We recommend considering the replacement of the mapping `proposalTimelocks` with a fixed length array in order to save gas during calls to functions `execute()` and `cancel()`.

Alleviation

`[Venus, 02/05/2024]`: "Issue acknowledged. I won't make any changes for the current version.

While we acknowledge this, we choose to abstain from making changes to uphold consistency across chains."

OPS-06 | UNNECESSARY CHECK

Category	Severity	Location	Status
Gas Optimization, Code Optimization	● Optimization	Cross-chain/OmnichainProposalSender.sol (Base-PR): 213~214	● Resolved

Description

Function `fallbackWithdraw()` includes the following check:

```
        require(address(this).balance >= originalValue_, "OmnichainProposalSender: insufficient native balance");
```

This check is unnecessary because it is checked that `keccak256(execution) == hash` , meaning that the `originalValue_` included as part of the `hash` construction was sent to the contract but currently has not successfully been transferred out.

Recommendation

We recommend removing this check from the `fallbackWithdraw()` function.

Alleviation

`[CertiK, 02/05/2024]` : The client made changes resolving the finding in commit e3d5e6606fb510bb0c09c4579af7beedb2cdb665.

TVG-02 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	● Optimization	TimelockV8.sol (Base-Timelock): 144, 145, 172, 173, 195, 196	● Resolved

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

Alleviation

`[CertiK, 02/05/2024]` : The client made changes resolving the finding in commit 54103aba18c7f5af7cf97c20a6312a81dee1635e.

APPENDIX | VENUS - MULTICHAIN GOVERNANCE

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 Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
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.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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.

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.