Quantstamp

Venus Multichain Support

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	DeFi Lending Protocol, Bridge
Timeline	2023-10-12 through 2023-10-27
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Protocol Documentation Whitepaper
Source Code	• VenusProtocol/venus-protocol #0a05857 • VenusProtocol/isolated-pools #5299454 • VenusProtocol/isolated-pools #5e660bf • VenusProtocol/oracle #a0a36bc • VenusProtocol/venus-protocol #a158f8c • VenusProtocol/protocol-reserve #e396119 • VenusProtocol/governance-contracts #358bed4
Auditors	• Shih-Hung Wang Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Julio Aguilar Auditing Engineer • Cameron Biniamow Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	30 Fixed: 15 Acknowledged: 13 Mitigated: 2
High severity findings	0
Medium severity findings	4 Fixed: 3 Acknowledged: 1
Low severity findings	18 Fixed: 9 Acknowledged: 7 Mitigated: 2
Undetermined severity findings	1 Acknowledged: 1
Informational findings	7 Fixed: 3 Acknowledged: 4

Summary of Findings

Venus Protocol is a DeFi lending protocol operating on the Binance Smart Chain. The focus of this audit was on the upgrades and modifications made to the protocol to extend support to additional EVM chains, specifically Ethereum, Arbitrum One, Polygon zkEVM, and opBNB. Additionally, a cross-chain bridge was developed to facilitate the transfer of XVS tokens, the governance token of the Venus Protocol, across these EVM chains.

The audit team discovered several issues related to the code compatibility with the above EVM chains and potential vulnerabilities in the XVS bridge. For example, the different or irregular block times on other EVM chains could cause inaccurate calculation of interest rates in the markets (VMC-5, VMC-6) and the distribution of XVS rewards to users (VMC-7, VMC-8). The sequencer downtime on L2 networks may affect the oracle usage (VMC-3) and access control mechanisms (VMC-29). Using a more recent Solidity version may cause the code to be incompatible with several EVM chains as well (VMC-9). Furthermore, the XVS bridge may accept messages from a compromised remote (VMC-1) or result in the lock of funds due to operational errors (VMC-2).

The code is well-written and has good documentation. However, the quality of the test suite could be improved. For some contracts within the scope of the audit, such as the XVS bridge contracts, branch coverage is only around 70%. Also, no tests are provided for the Treasury contract. We recommended to improve these values as close to 100% as possible.

The audit team has strictly covered the files in the Scope section, and any other files were out of the scope of this audit. It is strongly recommended the Venus team address all the issues outlined in this report.

Fix Review Update: All issues have been either fixed, mitigated, or acknowledged by the Venus team. Several contracts in the protocol were modified to support time-based interest rate calculation to mitigate issues VMC-5 and VMC-6. However, for issues VMC-7 and VMC-8, the XVS Vault is still using block-based reward calculation at the time of writing. For issue VMC-2, although it is resolved, it should be noted that the mitigation increases the centralization and operational risks of the protocol. See VMC-30 for more details.

ID	DESCRIPTION	SEVERITY	STATUS
VMC-1	XVS Bridges May Accept Messages from Compromised Remotes	Medium	Fixed
VMC-2	Potential of Locked Funds in XVS Bridges	Medium	Fixed
VMC-3	Missing Checks for Sequencer Uptime When Fetching Chainlink Prices	Medium	Fixed
VMC-4	Chainlink Oracle Can Have Its Prices Overridden	Medium	Acknowledged
VMC-5	Block Time Is Subject to Change	Low	Mitigated
VMC-6	Block Time on Polygon zkEVM Depends on Chain Usage	Low	Mitigated
VMC-7	Users Will Get Wrong Rewards Due to Arbitrum's `block.number` Incompatibility	Low	Acknowledged
VMC-8	Block Time Differences Between Chains Could Affect the Vault's Rewards	Low	Acknowledged
VMC-9	Potential EVM Compatibility Issue in Solidity Verison >=0.8.20	Low	Fixed
VMC-10	Use of Solidity's `transfer()` Function	Low	Fixed
VMC-11	Potential of Locked Funds in the XVS Bridge Admin Contract	Low	Fixed
VMC-12	XVS Bridges May Still Receive or Release Tokens When Paused	Low	Fixed
VMC-13	Inconsistent Data Format For Trusted Remotes in The XVS Bridge	Low	Fixed
VMC-14	Possible Failures of Bridging XVS Tokens	Low	Fixed
VMC-15	Extensive Reliance on External Applications	Low	Fixed
VMC-16	Oracle Cannot Handle ERC-20 Tokens with More than 18 Decimals	Low	Acknowledged
VMC-17	Renounceable Privileged Role	Low	Acknowledged
VMC-18	Newly Managed Contracts Do Not Restrict Privileged Accounts with Function-Level Access	Low	Acknowledged
VMC-19	Critical Role Transfer Not Following Two-Step Pattern	Low	Fixed
VMC-20	Missing Input Validation	Low	Fixed
VMC-21	Privileged Roles and Ownership	Low	Acknowledged
VMC-22	Avoid Setting Large Shared Decimals on XVS Bridges	Informational	Acknowledged
VMC-23	XVS Transfer Amount from Whitelisted Users Is Considered in Daily Limit	Informational	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
VMC-24	XVS Token's Name and Symbol Are Reversed	Informational	Fixed
VMC-25	Insufficient Validation in `isTrustedRemote()`	Informational	Fixed
VMC-26	Bypassable Events	Informational	Acknowledged
VMC-27	Use of Solidity Version with Known Compiler Bugs	Informational	Acknowledged
VMC-28	Unlocked Pragma	Informational	Acknowledged
VMC-29	Address Aliasing May Affect Cross-Chain Access Control	Undetermined	Acknowledged
VMC-30	Risks of Potential Operational Errors When Handling Failed Bridge Transfers	Low	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

- Venus Treasury V8: Repo `venus-protocol` at commit 0a05857:

    - `contracts/Governance/VTreasuryV8.sol`
- XVS Bridge: Repo `isolated-pools` at commit 5299454:
    - `contracts/Bridge/BaseXVSProxyOFT.sol`
    - `contracts/Bridge/XVSBridgeAdmin.sol`
    - `contracts/Bridge/XVSProxyOFTDest.sol`
    - `contracts/Bridge/XVSProxyOFTSrc.sol`
    - `contracts/Bridge/token/XVS.sol`
    - `contracts/Bridge/token/TokenController.sol`
    - `contracts/Bridge/interfaces/IXVS.sol`
    - `contracts/Bridge/interfaces/IXVSProxyOFT.sol`
- Isolated pool changes: Repo `isolated-pools` at commit 5e660bf:
    - `contracts/BaseJumpRateModelV2.sol`
    - `contracts/JumpRateModelV2.sol`
    - `contracts/WhitePaperInterestRateModel.sol`
    - `contracts/lib/constants.sol`
- Resilient Oracles changes: Repo `oracle` at commit a0a36bc:
    - `contracts/ResilientOracle.sol`
    - `contracts/oracles/ChainlinkOracle.sol`
- XVS Vault: `venus-protocol` at commit a158f8c:
    - `contracts/XVSVault/XVSStore.sol`
    - `contracts/XVSVault/XVSVault.sol`
    - `contracts/XVSVault/XVSVaultErrorReporter.sol`
    - `contracts/XVSVault/XVSVaultProxy.sol`
    - `contracts/XVSVault/XVSVaultStorage.sol`
- Protocol Share Reserve: `protocol-reserve` at commit e396119:
    - `contracts/ProtocolReserve/ProtocolShareReserve.sol`
- Access Control Manager: `governance-contracts` at commit 358bed4:
    - `contracts/Governance/AccessControlManager.sol`

At the time of the fix review, the XVS Bridge contracts have been moved to a separate repo:

- XVS Bridge: Repo `token-bridge`, the latest reviewed commits are 06c6009 and 6229b5e:
    - `contracts/Bridge/BaseXVSProxyOFT.sol`
    - `contracts/Bridge/XVSBridgeAdmin.sol`
    - `contracts/Bridge/XVSProxyOFTDest.sol`
    - `contracts/Bridge/XVSProxyOFTSrc.sol`
    - `contracts/Bridge/token/XVS.sol`
    - `contracts/Bridge/token/TokenController.sol`
    - `contracts/Bridge/interfaces/IXVS.sol`
    - `contracts/Bridge/interfaces/IXVSProxyOFT.sol`

Findings

VMC-1

XVS Bridges May Accept Messages from Compromised Remotes

Medium Fixed

Update

The `BaseXVSProxyOFT` contract now overrides `_nonblockingLzReceive()` and includes logic that validates that `_srcAddress` is the trusted remote for the given `_srcChainId`. Further, the `removeTrustedRemote()` function was created to explicitly update the `trustedRemoteLookup` mapping to `0x0` for a given remote chain ID.

Update

Marked as "Fixed" by the Venus team. Addressed in: `a5eb339dbdf9d98916f33c41ed3b2a50c5531d2e`, `76c32e99317930912f82964d19cbd4e53b977f86`.

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The XVS bridges are set up as non-blocking LayerZero apps, which allows for the replaying of unsuccessful messages via `NonblockingLzApp.retryMessage()`. `retryMessage()` makes a subsequent call to `OFTCoreV2._nonblockingLzReceive()`, which attempts to retry the failed message. This can be an issue if a trusted remote is compromised and is removed as a trusted remote since neither `retryMessage()` nor `_nonblockingLzReceive()` perform any validation checks on whether the `_srcAddress` on the source chain is still a trusted remote.

Exploit Scenario:

1. Contract X on Chain A is set up as a trusted remote.
2. Several messages from Contract X fail during execution and are stored in the `failedMessages` mapping on Chain B.
3. On Chain B, the owner/multi-sig removes Contract X as a trusted remote of Chain A.

4. A malicious actor can still call `retryMessage()`, potentially executing any failed messages that originated from the invalidated Contract X.

Recommendation: Confirm whether this is an intended protocol design. If not, consider adding additional validation on `_srcAddress`, similar to what is performed in `LzApp.lzReceive()`, by overriding the `retryMessage()` function.

VMC-2 Potential of Locked Funds in XVS Bridges

Medium Fixed

Update

1st Fix Review: Both `XVSProxyOFTSrc` and `XVSProxyOFTDest` contracts include a `dropFailedMessage()` function to drop any failed cross-chain messages. A `fallbackWithdraw()` function is added to `XVSProxyOFTSrc` to release the locked funds in the source bridge after the failed message in the destination bridge is dropped. However, if the `sendAndCall()` function is used, and the callback function fails, the newly minted XVS will be locked in the destination bridge.

2nd Fix Review: A `sweepToken()` function is added to recover locked tokens (not limited to only XVS) in the bridge contracts. An admin-controlled flag is added to allow the admin to activate or deactivate the `sendAndCall()` function. Although the issue of the potential of locked funds is resolved, it should be noted that the mitigation increases the centralization and operational risks of the protocol. See VMC-30 for more details.

Update

Marked as "Fixed" by the Venus team. Addressed in: `264109ce597e95b4ea1a630e7c68dada143968fe` (isolated-pools repo), `6229b5e6082608fef2215cb7cc5f5f6588d79df9` (token-bridge repo), `06c6009e01411182d738b2249cbbb06019926b54` (token-bridge repo).

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The function `BaseOFTV2.sendAndCall()` burns or locks XVS tokens from users with the intention of bridging them over to another chain to an address provided by the user. However, the recipient, `_toAddress`, **must** be a contract on the destination chain. Otherwise, the function `OFTCoreV2._sendAndCallAck()` will mint or release tokens through `_creditTo()` without transferring them to the designated address, essentially locking the XVS tokens in the bridge contract.

Additionally, if for some reason a failed transaction is unable to be retried through `retryMessage()`, tokens might be unable to be sent to the destination address even though they might have been burned in the source chain.

Recommendation: This issue originates from the design of LayerZero's `OFTCoreV2` contract. Funds locked in the bridge should be avoided. Since the source bridge cannot verify that the destination address is a contract, we recommend introducing a mechanism that can return the locked funds in the source chain.

VMC-3 Missing Checks for Sequencer Uptime When Fetching Chainlink Prices

Medium Fixed

Update

Marked as "Fixed" by the Venus team. Addressed in: `5e2dcbf33e92fe0865134653f87779f06f563083`.

File(s) affected: `oracle/contracts/oracles/ChainlinkOracle.sol`

Description: The `ChainlinkOracle._getChainlinkPrice()` function fetches the latest price of an asset from Chainlink price feeds. However, as the protocol is intended to be deployed on Arbitrum and opBNB, checking the sequencer's uptime before querying price data is crucial. If the sequencer is down, there is a chance of using an incorrect or stale price. Please refer to Chainlink L2 Sequencer Uptime Feeds document for more details.

Recommendation: On Arbitrum, consider checking the sequencer's uptime feed to avoid using stale prices. An example code can be found in the Chainlink official documentation.

VMC-4 Chainlink Oracle Can Have Its Prices Overridden

Medium Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> The current behavior is the expected one. Direct prices are set via VIP (Venus Improvement Proposal), with the approval of the Venus Community. And it's expected they have a higher priority than other sources, in any case

File(s) affected: `oracle/contracts/oracles/ChainlinkOracle.sol`

Description: `setDirectPrice()` allows a user with sufficient access control to manually set a price for any asset in the `prices[]` mapping. `_getPriceInternal()` will return the `prices[]` mapping if it is non-zero, preferring it over the price returned by `_getChainlinkPrice()`.

The comment specifies that the manually set price is "useful under extenuating conditions such as price feed failure". However, the price is currently used in preference of the price feed, even if it is successful. Additionally, the price never becomes stale. Even if the price is accurately set on initialization, it may not remain so and the protocol would continue operating under a false price.

Recommendation: Programmatically check for a price feed failure before returning the manual price. Check whether or not the manual price has become stale.

VMC-5 Block Time Is Subject to Change

• Low ⓘ Mitigated

Update

The Venus team has mitigated this issue:

- A new contract, `TimeManager`, is introduced to allow the admin to configure the base interest rates based on per block or second. The `JumpRateModelV2` and the `WhitePaperInterestRateModel` contracts are now inherited from `TimeManager`. However, only part of the code variable names and comments reflect the change.
- The `TimeManager.blocksOrSecondsPerYear` is set to immutable. Therefore, the contracts have to be redeployed if the chain's block time changes in the future, as mentioned by the Venus team.
- Note that the `VToken` and other related contracts have been updated to reflect the change. However, those contracts are out of the scope of this audit. Please refer to the Scope section for a complete list of in-scope files.

Update

Marked as "Fixed" by the Venus team. Addressed in: `92e8e7555e6e0b468c51bd30ebf1ee2045a6877f`. The Venus team provided the following explanation:

> We will be using block.timestamp instead of block.number, and “seconds per year” instead of “blocks per year”, for the L2’s. This will be achieved configuring in deployment time the new `TimeManager` abstract contract, that will be used by every contract in the protocol that depended on the block number in the past. For Ethereum mainnet, if the blocks per year change in a significant way in the future (it seems reasonable), the current plan is to upgrade the implementations with an updated value of the immutable variable `blocksOrSecondsPerYear`.

File(s) affected: `isolated-pools/contracts/WhitePaperInterestRateModel.sol`, `isolated-pools/contracts/BaseJumpRateModelV2.sol`, `venus-protocol/contracts/XVSVault/`

Description: The interest rate contracts store an immutable variable `blocksPerYear` to help with calculations. However, the block time for each chain is not guaranteed to remain the same across L2's. For opBNB, the currently configured block time is 1 second, according to the official document, which, however, includes a warning that it is susceptible to changes.

Recommendation: Update `blocksPerYear` to a mutable variable to handle changes in chain configuration. In the current design, `BaseJumpRateModelV2` and the XVS vault allow a privileged role to adjust the parameters, while `WhitePaperInterestRateModel` doesn't. Alternatively, have a migration plan for the event that block times change.

VMC-6 Block Time on Polygon zkEVM Depends on Chain Usage

• Low ⓘ Mitigated

Update

The Venus team has mitigated this issue. See VMC-5 for more details.

Update

Marked as "Fixed" by the the Venus team. Addressed in: `92e8e7555e6e0b468c51bd30ebf1ee2045a6877f`. The Venus team provided the following explanation:

> We will be using block.timestamp instead of block.number for the L2’s. See explanation in VMC-5

File(s) affected: `isolated-pools/contracts/WhitePaperInterestRateModel.sol`, `isolated-pools/contracts/BaseJumpRateModelV2.sol`, `venus-protocol/contracts/XVSVault/`

Description: The protocol's interest rate models calculate the borrow and supply rates per block of the markets based on their current utilization rate. The `blocksPerYear` immutable variable in the models is configured based on the deployed chain to adjust the market APR.

On Ethereum, BSC, and opBNB chains, the block time can be considered a reliable source of timing information. However, for Polygon zkEVM, the block time is irregular at the time of writing and depends on the chain usage. According to the historical average block time on the Polygon zkEVM explorer, we can observe that the average block time varies over time and is relatively unreliable.

This is because, currently, Polygon zkEVM blocks are produced on-demand, according to rollup.codes,

> New L2 blocks contain only one L2 transaction and is created on-demand, i.e., once there is a new transaction.

An irregular block time may also affect the reward distribution rate of the XVS vault, which calculates the reward amount based on a configured `rewardTokenAmountsPerBlock` value and the number of passed blocks.

Recommendation: Consider clarifying the risks of a variable block time on Polygon zkEVM could cause to the users on public-facing documentation. Using the block timestamp for timing can be an alternative, as it is more reliable and restricted by the L1 timestamp.

VMC-7
Users Will Get Wrong Rewards Due to Arbitrum's `block.number` Incompatibility

• Low ⓘ Acknowledged

Update

This issue is marked as Acknowledged. See VMC-8 for more details.

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> We will be using block.timestamp instead of block.number for the L2's. See explanation in VMC-8

File(s) affected: `venus-protocol/contracts/XVSVault/`

Description: According to Arbitrum's documentation, its `block.number` is updated to sync with Ethereum's `block.number` every minute. Therefore, any time calculations based on the block number would be inaccurate.

The `XVSVault` relies on the `block.number` to update `pool.accRewardPerShare` in `_updatePool()` based on the duration since the last update. The pool is updated internally before every major action is taken in the contract, and there is also the external function `updatePool()` without any restrictions on the caller. If `_updatePool()` is called less frequently, this could cause inaccuracies in `accRewardPerShare` in the short term, leading to users receiving fewer rewards than expected.

On the other hand, block numbers and timestamps on Arbitrum are generally reliable in the longer term, according to the documentation:

> As a general rule, any timing assumptions a contract makes about block numbers and timestamps should be considered generally reliable in the longer term (i.e., on the order of at least several hours) but unreliable in the shorter term (minutes). (It so happens these are generally the same assumptions one should operate under when using block numbers directly on Ethereum!)

If the rewards need to be accurate on the order of several hours, this issue is of less concern.

Recommendation: Depending on the needs of the contract, consider whether the short-term inaccuracies are acceptable. If not, one possible solution could be to use Arbitrum's precompile `AbsSys.arbBlockNumber()`, which returns the current L2 block number. However, the block time in Arbitrum (current average is 0.26 seconds) is different, and additional adjustments may be required to be fully equivalent to how it works in the Binance chain (3 seconds), see issue VMC-8. Also, L2 block time on Arbitrum depends on chain usage and therefore is considered irregular.

VMC-8
Block Time Differences Between Chains Could Affect the Vault's Rewards

• Low ⓘ Acknowledged

Update

This issue is marked as Acknowledged as the reward calculation in the XVS Vault contracts still relies on `block.number`.

> **Update**
> Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:
>
> > We will be using block.timestamp instead of block.number for the L2's. The rewards will be set according to that in the L2's. We'll develop a TimeManager contract compatible with solidity 0.5.16
> >
> > Moreover, rewards in the XVSVault will be set via VIP (Venus Improvement Proposal), involving the Venus Community, that will be able to review it carefully before the execution

File(s) affected: `venus-protocol/contracts/XVSVault/`

Description: The XVS vault internal mapping `rewardTokenAmountsPerBlock`, as the name implies, stores the amount of rewards per block for each registered token. On chains with a fixed block time, since each block is created (approximately) periodically, this implies "per unit of time". Therefore, as seen in the table below, special care needs to be taken when setting this value in the different chains (these values can and might change in the future). A wrong value could result in unexpected rewards being provided to users. For chains with irregular block times, please refer to the other related issues.

Chain	Avg Blocktime [s]
Ethereum	12
BSC	3
Optimism	2
opBNB	1
Arbitrum	~12 (L1), Irregular (L2)
zkEVM	Irregular

References:

- opBNB vs other Layer2 Networks | BNB Optimistic Rollup
- Arbitrum Block Numbers and Time

Recommendation: Make sure to understand the consequences the block time has over the rewards rate and set `rewardTokenAmountsPerBlock` accordingly in the different chains.

VMC-9
Potential EVM Compatibility Issue in Solidity Verison >=0.8.20

• Low Fixed

> **Update**
> The contract is compiled under Solidity version 0.8.20 with the EVM version set to `paris` in the Hardhat config file.

> **Update**
> Marked as "Fixed" by the Venus team. Addressed in: `ca7bb135c9e182ac736e4ac8222699d68288d9a2`.

File(s) affected: `venus-protocol/contracts/Governance/VTreasuryV8.sol`

Description: According to the Hardhat config file, the `VTreasuryV8` contract is compiled in Solidity version 0.8.20 with the EVM version set as the default Shanghai version. As a result, the contract bytecode may include `PUSH0` opcodes, which are not supported on certain EVM chains, including BSC, Arbitrum, Optimism, and opBNB. The deployment or execution of the contract may fail due to the use of an invalid opcode.

Recommendation: Consider explicitly setting the EVM version as Paris in the Hardhat config file to prevent the compiler from generating `PUSH0` opcodes in the bytecode. Alternatively, consider using a Solidity version of 0.8.19.

VMC-10 Use of Solidity's `transfer()` Function

• Low Fixed

> **Update**
> The code no longer uses `transfer()` but changes to `address.call()`. Also, a reentrancy guard is added to both withdrawal functions.

> **Update**
> Marked as "Fixed" by the Venus team. Addressed in: `818631f05e33e52e85df9c1a5d4d1c4bef503ad5`, `ca7bb135c9e182ac736e4ac8222699d68288d9a2`.

File(s) affected: `venus-protocol/contracts/Governance/VTreasuryV8.sol`

Description: Solidity's `transfer()` function forwards a fixed amount of 2300 gas to the recipient when transferring native tokens to them, which causes several limitations if the recipient is a contract. First, the limited amount of gas prevents the contract from executing more complicated logic (e.g., external calls) in its `fallback()` or `receive()` function. Second, the gas costs of each opcode are subject to change, and therefore, it is not guaranteed that currently successful transfers will not fail in the future.

The `VTreasury.withdrawTreasuryNative()` allows the owner to withdraw native tokens from the Venus Treasury, which uses Solidity's `transfer()` function to transfer native tokens to a designated address.

Recommendation: Consider using `.call{value: value_}("")` to transfer native tokens to the recipient. It should be noted that this approach forwards all the remaining gas to the recipient and may open an attack vector of reentrancy. Therefore, additional measures such as following the CEI pattern (which is implemented currently) or adding a reentrancy guard on the functions are suggested.

VMC-11
Potential of Locked Funds in the XVS Bridge Admin Contract

• Low Fixed

Update

Fixed by removing the `payable` keyword from the `fallback()` function.

Update

Marked as "Fixed" by the Venus team. Addressed in: `da84da2e60cd4331528eb646d6ccb6d0b2ee575c` .

File(s) affected: `isolated-pools/contracts/Bridge/XVSBridgeAdmin.sol`

Description: The `XVSBridgeAdmin` contract has a `payable` fallback function without a withdraw function. Also, there is no native token provided in the call to the XVS bridge contract. As a result, any native tokens sent in a fallback function call will be locked in the admin contract.

Recommendation: Depending on the protocol design, consider the following mitigations:

1. If some function call to the XVS bridge may require native tokens, consider forwarding `msg.value` in the call.
2. If no native token is needed for any call to the XVS bridge, consider removing the `payable` keyword to avoid the unexpected lock of native tokens in the admin contract.
3. If the XVS bridge or any other contract may transfer native tokens to the admin contract, consider adding an empty `receive()` function.

VMC-12
XVS Bridges May Still Receive or Release Tokens When Paused

• Low Fixed

Update

The Venus team has fixed this issue:

- Scenario 1 is clarified in the XVS bridge documentation, Section "6.2. Pause and Resume."
- Scenario 2 is fixed by adding a `whenNotPaused` modifier to `BaseXVSProxyOFT._transferFrom()` to prevent transfers.

Update

Marked as "Fixed" by the Venus team. Addressed in: `63b4e553212d0f42169f6c75737a5018a283254d` . The Venus team provided the following explanation:

> We won't allow the execution of the `_transferFrom` function in the `BaseXVSProxyOFT` when the contract is paused. This way, the second scenario described will be mitigated.
>
> Regarding the first scenario described, we'll explain it in our public documentation site: https://github.com/VenusProtocol/venus-protocol-documentation/pull/77/commits/244a44662f105112bc61872c80fc890e6e4d4574

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The `owner` of the XVS bridge can pause or unpause the bridge. When the bridge is paused, function calls to `_debitFrom()` and `_creditTo()` will revert. The `_debitFrom()` function is called when a user initiates a transfer on the source chain, and the `_creditTo()` function is called when the user receives the transferred token on the destination chain.

No user can initiate a transfer from the source bridge to others when a source bridge is paused. However, when a destination bridge is paused, the following scenarios are still possible:

1. The destination bridge can receive cross-chain messages, which attempt to mint or release tokens to the receiver. Although these messages will fail, they can be retried once the destination bridge is unpaused.
2. Messages of type `PT_SEND_AND_CALL` will trigger an `onOFTReceived()` function call to the recipient. If the call fails, the messages will be stored, but the XVS tokens will be credited to the bridge. These messages can be retried during the pause period since they do not

execute the `_creditTo()` but the `_transferFrom()` function. As a result, the recipient will still be able to receive the tokens during the pausing period.

Recommendation: Confirm whether this is an intended protocol design. If so, clarify that minting or releasing funds may still happen when a bridge is paused in public-facing documents.

VMC-13
Inconsistent Data Format For Trusted Remotes in The XVS Bridge

Low Fixed

Update

Marked as "Fixed" by the Venus team. Addressed in: `35a3312659c8b0f443c6ded33e613aed7130d2e2` , `cb84e01596d687c87c26c2ed16f24040965cad32` . The Venus team provided the following explanation:

> We are not granting permissions in ACM to deployer/timelock for `setTrustedRemote` . This omission of permissions will ensure the use of `setTrustedRemoteAddress` function only.
>
> `removeTrustedRemote` added in `cb84e01596d687c87c26c2ed16f24040965cad32`

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The owner of the bridge can set and update trusted remote addresses to verify that interactions with LayerZero are generated from trusted contracts on remote chains. Trusted remote addresses are stored in the mapping `trustedRemoteLookup` where the key is the remote chain ID and the data stored is bytes in the format of `abi.encode(<REMOTE_ADDRESS>, <LOCAL_ADDRESS>)` .

Setting trusted remote addresses is performed by calling `setTrustedRemoteAddress()` where the remote address is passed as an argument and then encoded in the aforementioned format.

Similarly, `setTrustedRemote()` allows for setting a trusted remote with data encoded off-chain. Since `setTrustedRemote()` does not validate the input data, it is entirely possible to encode the incorrect data or encode it in the incorrect format.

If the `trustedRemoteLookup` mapping does not contain data in the correct format, bridging tokens to another chain will fail.

Recommendation: Unfortunately, the functions in question are not virtual so they cannot be overridden. Since there is another issue VMC-25 related to this contract, consider creating a new contract where only `setTrustedRemoteAddress()` is to be used, and `setTrustedRemote()` is replaced by a new function to explicitly remove trusted remote addresses, for example `removeTrustedRemote(uint16 _remoteChainId)` .

VMC-14 Possible Failures of Bridging XVS Tokens

Low Fixed

Update

Marked as "Fixed" by the Venus team. Addressed in: `244a44662f105112bc61872c80fc890e6e4d4574` . The Venus team provided the following explanation:

> The commit is actually in the documentation repo: https://github.com/VenusProtocol/venus-protocol-documentation/commit/244a44662f105112bc61872c80fc890e6e4d4574

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: This issue highlights the potential failures users may encounter when bridging the XVS tokens. The following scenario may prevent users from sending XVS tokens from the source chain:

1. The oracle temporarily fails due to reasons including being paused by the owner, incorrect address configuration, or price validation failures.
2. The transfer amount exceeds the single or daily sending transaction limit.
3. The transfer amount is too small, becoming zero after removing dust.
4. The sender is blacklisted by the XVS token.
5. The destination bridge is not configured as a trusted remote.

The following scenario may prevent users from receiving XVS tokens on the destination chain. Users may retry the failed messages after some period:

1. The oracle temporarily fails due to reasons including being paused by the owner, incorrect address configuration, or price validation failures.
2. The transfer amount exceeds the single or daily receiving transaction limit.
3. The recipient is blacklisted by the XVS token.
4. The minting cap on the destination bridge is exceeded.

Specifically, if the source bridge is not configured as a trusted remote on the destination chain, the cross-chain messages from the source bridge will be blocked. A privileged role has to add or update the trusted remotes on the destination chain and call `retryPayload()` on LayerZero's Endpoint to retry the blocked message.

Note that the `LzApp.forceResumeReceive()` function will drop the blocked message without retrying forwarding it to the destination bridge, which should only be used in extreme circumstances as it can potentially cause loss of funds.

Recommendation: Consider clarifying the potential failures of the bridging process to users in public-facing documentation, including guidelines for users to follow when they fail to send or receive tokens. Also, consider if it is necessary to provide a way for users to claim a refund on the source chain in case the failure on the destination chain is persistent and irrecoverable.

VMC-15 Extensive Reliance on External Applications

Low Fixed

Update

Marked as "Fixed" by the Venus team. Addressed in: `244a44662f105112bc61872c80fc890e6e4d4574`. The Venus team provided the following explanation:

> The commit is actually in the documentation repo: https://github.com/VenusProtocol/venus-protocol-documentation/commit/244a44662f105112bc61872c80fc890e6e4d4574

File(s) affected: `isolated-pools/contracts/Bridge/`

Description: The protocol depends on LayerZero for bridging XVS tokens to other chains. As such, the token on each network can be impacted by the possible vulnerabilities in the bridging mechanism. This is an inherent risk of integrating with network bridges, not a specific vulnerability of the token implementation.

Recommendation: We recommend communicating this feature and its corresponding considerations to the community in public-facing documentation. In the future, consider implementing alternative bridging mechanisms and allow users to choose between them.

VMC-16 Oracle Cannot Handle ERC-20 Tokens with More than 18 Decimals

Low Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> We have an internal guideline with rules to be applied to new underlying tokens. Venus protocol doesn't support underlying tokens with more than 18 decimals.

File(s) affected: `oracle/contracts/oracles/ChainlinkOracle.sol`

Description: The `ChainlinkOracle.getPrice()` and `_getPriceInternal()` functions fetch the given ERC-20 token's `decimal` and multiply the asset price by `10 ** (18 - decimal)`. However, this approach cannot handle ERC-20 tokens with decimals larger than 18, as the subtraction will trigger an integer overflow.

Recommendation: If the protocol possibly supports ERC-20 tokens with decimals larger than 18, consider refactoring the `_getPriceInternal()` to handle such a case.

VMC-17 Renounceable Privileged Role

Low Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> `DEFAULT_ADMIN_ROLE` will be granted to Normal Timelock as we did here https://bscscan.com/tx/0×66b32b0d8918b43e43e2b6104927273f012b81ad8ee30d1284c6067aa761b687
>
> Normal Timelock is the contract to execute Normal VIP (Venus Improvement Proposals). So, the renouncement should have to be done with the votes of the community, which is unlikely.

File(s) affected: `governance-contracts/contracts/Governance/AccessControlManager.sol`

Description: In the contract `AccessControlManager`, the `DEFAULT_ADMIN_ROLE` is granted to the deployer, which can then grant and revoke privileged roles for access to restricted protocol functions. In the case where the deployer calls `AccessControlManager.renounceRole(DEFAULT_ADMIN_ROLE, deployerAddress)`, and no other addresses are granted the `DEFAULT_ADMIN_ROLE`, the `AccessControlManager` contract will no longer be able to grant or revoke roles. In certian cases, a renounced `DEFAULT_ADMIN_ROLE` could prevent a managed contract from accessing pausable functionality or other critical functionality.

Recommendation: Consider replacing the inherited contract `AccessControl` with `AccessControlDefaultAdminRules` from OpenZeppelin, which contains additional protections for the `DEFAULT_ADMIN_ROLE`, such as following a two-step transfer process when transferring or renouncing the role.

VMC-18

Newly Managed Contracts Do Not Restrict Privileged Accounts with Function-Level Access

Low Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> We avoid as much as possible the use of the `ZERO_ADDRESS` in the `AccessControlManager`. It's only used to grant permissions on specific functions of the `VToken` contracts (there are several `VToken` contracts, so this way we can save gas setting permissions).

File(s) affected: `governance-contracts/contracts/Governance/AccessControlManager.sol`

Description: The contract `AccessControlManager` allows the `DEFAULT_ADMIN_ROLE` to grant an account access to a specified function accross **all** managed contracts by passing the zero address when calling `AccessControlManager.giveCallPermission()`. While code comments indicate that this is the intended design, it is crucial to note that such a design *could* lead to uninteded access when a new contract is managed by the `AccessControlManager`, or when a currently managed contract is upgraded with new functions.

Exploit Scenario: Assume that the contracts `A` and `B` are managed by the `AccessControlManager`.

```
contract A {
    function restricted() external {
        require(
            AccessControlManager(0x...).isAllowedToCall(
                msg.sender,
                "restricted()"
            ),
            "..."
        );

        ...
    }
}

contract B {
    function restricted() external {
        require(
            AccessControlManager(0x...).isAllowedToCall(
                msg.sender,
                "restricted()"
            ),
            "..."
        );

        ...
    }
}
```

1. The `DEFAULT_ADMIN_ROLE` calls `AccessControlManager.giveCallPermission(address(0), "restricted()", 0x1234...)`, which allows the address `0x1234...` to call the function `restricted()` on both contracts `A` and `B`.
2. A new contract, `C`, is managed by the `AccessControlManager` where the `restricted()` function is only intended to be called by the address `0x9876...`.

```
contract C {
    function restricted() external {
        require(
            AccessControlManager(0x...).isAllowedToCall(
                msg.sender,
                "restricted()"
            ),
```

```
                "..."
            );

            ...
        }
    }
```

3. The `DEFAULT_ADMIN_ROLE` calls `AccessControlManager.giveCallPermission(contractCAddr, "restricted()", 0x9876...)`, which allows the address `0x9876...` to call the function `restricted()` on **only** the contract `C`.
4. The address `0x1234...` successfully calls the restricted function `C.restricted()`.

Recommendation: It is crucial that the Venus team verifies that new contracts managed by the `AccessControlManager` contract do not contain functions that could be accessed by undersired accounts.

VMC-19 Critical Role Transfer Not Following Two-Step Pattern

Low Fixed

Update

This issue is fixed by updating the `Ownable` dependency to `Ownable2Step`. However, the override of the `renounceOwnership()` function has been removed, causing the contract to be renounceable by the owner.

Update

Marked as "Fixed" by the Venus team. Addressed in: `8a2da0f2ede2cadc981531d5e226ac6aac558342`.

File(s) affected: `venus-protocol/contracts/Governance/VTreasuryV8.sol`, `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The owner of the contracts can call `transferOwnership()` to transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new owner address then the contract will no longer have an active owner, and functions with the `onlyOwner` modifier can no longer be executed.

Recommendation: Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern in which the new owner must accept their position before the transfer is complete.

VMC-20 Missing Input Validation

Low Fixed

Update

Marked as "Fixed" by the Venus team. Addressed in: `0c5f140989ee41e6ba0e2d165596d8cac1005686`, `83ceb365ba6ca55997be0e925c0b8ffa755ac626`. The Venus team provided the following explanation:

> Oracle : It could be zero see here https://github.com/VenusProtocol/oracle/pull/124/files#diff-e830326f60f1d3c44c1d96d80684d24b8a35af7aa7a7f66d5de597917c599b50R123

File(s) affected: `venus-protocol/contracts/Governance/VTreasuryV8.sol`, `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`, `oracle/contracts/ResilientOracle.sol`

Description:

- `VTreasuryV8.withdrawTreasuryToken()`:
  - Validate that `tokenAddress` and `withdrawAddress` are not `address(0)` to avoid losing funds.
  - Validate that `withdrawAmount` is not 0.
- `VTreasuryV8.withdrawTreasuryNative()`:
  - Validate that `withdrawAddress` is not `address(0)` to avoid loss of funds.
  - Validate that `withdrawAmount` is not 0.
- `BaseXVSProxyOFT.setMaxSingleTransactionLimit()`:
  - Validate that the `limit_` is less than or equal to the `chainIdToMaxDailyLimit`.
- `BaseXVSProxyOFT.setMaxSingleReceiveTransactionLimit()`:
  - Validate that the `limit_` is less than or equal to the `chainIdToMaxDailyReceiveLimit`.
- `ResilientOracle.constructor()`:
  - Validate that `nativeMarketAddress` is not the zero address.

Recommendation: Consider adding the validation checks in the above functions.

VMC-21 Privileged Roles and Ownership

Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:
>
> > The owner of the Venus contracts is Normal Timelock on the BNB chain. Initially, on Ethereum, it will be a multisig wallet, but later in phase 2, it will be the Normal Timelock too, and all actions will be carried out via VIP only.

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. In this protocol, access control is also granted on a per-function basis through `_checkAccessAllowed()`.

VTreasuryV8

The `VTreasuryV8.sol` contract contains the following privileged roles:

1. An owner (`owner`, `onlyOwner` modifier) as initialized during the constructor execution to `msg.sender`:
    1. Transfer any amount of any token to any address by calling `withdrawTreasuryToken()`.
    2. Transfer any amount of the native token to any address by calling `withdrawTreasuryNative()`.

XVS Bridge & XVS

The `BaseXVSProxyOFT.sol` contract contains the following privileged roles:

1. An owner (`owner`, `onlyOwner` modifier) as initialized during the constructor execution to `msg.sender`:
    1. `setOracle()`: Set the address of the `ResilientOracle` contract.
    2. `setMaxSingleTransactionLimit()`: Sets the limit of a single transaction amount.
    3. `setMaxDailyLimit()`: Sets the limit of daily (24-hour) transactions amount.
    4. `setMaxSingleReceiveTransactionLimit()`: Sets the maximum limit for a single receive transaction.
    5. `setMaxDailyReceiveLimit()`: Sets the maximum daily limit for receiving transactions.
    6. `setWhitelist()`: Sets the whitelist address to skip checks on transaction limit.
    7. `pause()`: Pauses the bridge.
    8. `unpause()`: Unpauses the bridge.

The `XVSBridgeAdmin.sol` contract contains the following privileged roles:

1. An owner (`owner`, `onlyOwner` modifier) as initialized during the constructor execution to `msg.sender`:
    1. Insert or modify the mapping between function signatures and names to allow specific roles to call the privileged functions on the bridges.

The `XVS.sol` contract contains the following privileged roles:

1. An Access Control Manager (`accessControlManager_`), as initialized during the constructor execution to `accessControlManager_`:
    1. Control who is allowed to mint tokens by indirectly affecting `_ensureAllowed("mint(address,uint256)")`.
    2. Control who is allowed to burn tokens by indirectly affecting `_ensureAllowed("burn(address,uint256)")`.

The following functions in this contract are protected through `_ensureAllowed()`:

1. `mint(address,uint256)`: Mints a capped `amount_` of tokens to a specified `account_`.
2. `burn(address,uint256)`: Burns any `amount_` of tokens from a specified `account_`.

XVS Vault

The `XVSVault.sol` contract contains the following privileged roles:

1. An administrator (`admin`, `onlyAdmin()` modifier), as initialized during the constructor execution to `msg.sender`:
    1. Change the XVS store contract address (`xvsStore`) that controls reward token transfers by calling `setXvsStore()`.
    2. Change the access control manager implementation address (and thereby indirectly all `_checkAccessAllowed()`-controlled access checks) by calling `setAccessControl()`.

The following functions in this contract are protected through `_checkAccessAllowed()` with their corresponding function signatures:

1. `pause()`: Pause the contract (and thereby all calls to `deposit()`, `claim()`, `executeWithdrawal()`, `requestWithdrawal()`, `updatePool()`, `delegate()` and `delegateBySig()`) by calling `pause()`.
2. `resume()`: Resume the contract from a paused state by calling `resume()`.
3. `add(address,uint256,address,uint256,uint256)`: Add a new pool by calling `add()`.
    1. Using an existing reward token, but a new pool token allows the caller to globally update the `rewardTokenAmountsPerBlock[]` variable for all pools using this reward token, **without explicitly calling** `setRewardAmountPerBlock()`. In extreme cases, this could either drain the reward token (by choosing a high reward value) or halt any future rewards (by choosing zero).
    2. **Accidentally or with malicious intent the caller could add enough new pools to make calling** `massUpdatePools()` **prohibitively gas-expensive and thereby deny the service to the following functions**:
        1. `add()`
        2. `set()`
        3. `setRewardAmountPerBlock()`
4. `set(address,uint256,uint256)`: Update the accumulated rewards per share for all pools and change the allocation point of a pool by calling `set()`.

5. `setRewardAmountPerBlock(address,uint256)` : Update the accumulated rewards per share for all pools and arbitrarily change the reward amount per block by calling `setRewardAmountPerBlock()` .
    1. **By setting it to zero no additional rewards would be accumulated.**
    2. **By setting it to an unusually high value the reward token could be drained.**
6. `setWithdrawalLockingPeriod(address,uint256,uint256)` : Change the lockup period for withdrawing deposited funds by calling `setWithdrawalLockingPeriod()` .

The `XVSStore.sol` contract contains the following privileged roles:

1. An administrator (`admin` , `onlyAdmin()` modifier), as initialized during the constructor execution to `msg.sender` :
    1. Transfer the role to another address (which has to accept it by calling `acceptAdmin()`) by calling `setPendingAdmin()` .
    2. Designate a new owner role by calling `setNewOwner()` .
    3. Add/remove token addresses from the `rewardTokens[]` array by calling `setRewardToken()` .
2. An owner (`owner` , `onlyOwner()` modifier), as set through `setNewOwner()` by `admin` :
    1. Transfer an arbitrary amount of reward tokens by calling `safeRewardTransfer()` .
        1. **A compromised owner could drain all funds from this contract by**:
            1. Adding a to-be-drained token to the `rewardTokens[]` array, if not already a reward token.
            2. Transfer an arbitrary amount by calling `safeRewardTransfer()` .
        2. **A compromised admin could drain all funds from this contract by**:
            1. Adding a to-be-drained token to the `rewardTokens[]` array, if not already a reward token.
            2. Set themselves as an owner by calling `setNewOwner()` .
            3. Transfer an arbitrary amount by calling `safeRewardTransfer()` .
    2. The owner can drain this contract of any ERC-20 token held within it by calling `XVSVault.emergencyRewardWithdraw()` .

Resilient and Chainlink Oracles

The following functions in `ResilientOracle.sol` are protected through `_checkAccessAllowed()` :

1. `pause()` : Pauses the oracle.
2. `unpause()` : Unpauses the oracle.
3. `setOracle(address,address,uint8)` : Sets the address for the main, pivot, or fallback oracle.
4. `enableOracle(address,uint8,bool)` : Enables the main, pivot, or fallback oracle.
5. `setTokenConfig(TokenConfig)` : Assigns an asset to an oracle.

The following functions in `ChainlinkOracle.sol` are protected through `_checkAccessAllowed()` :

1. `setDirectPrice(address,uint256)` : Manually sets the price for an asset. **The manual price overrides any prices returned by the chainlink oracle, even if the oracle is functioning correctly.**
2. `setTokenConfig(TokenConfig)` : Assigns an asset to a price feed and defines a maximum stale period.

Protocol Share Reserves

The `ProtocolShareReserves.sol` contract contains the following privileged roles:

1. An owner (`owner` , `onlyOwner` modifier) as initialized during the constructor execution to `msg.sender` :
    1. `setPrime()` : Setter for the prime contract.
    2. `setPoolRegistry()` : Setter for the pool registry.

The following functions in this contract are protected through `_ensureAllowed()` :

1. `addOrRemoveAssetFromPrime(address,bool)` : Updates local storage to reflect whether an asset is stored in the Prime contract.
2. `addOrUpdateDistributionConfigs(DistributionConfig[])` : Adds/Updates the distribution between different distribution targets.
    This can add beneficiaries when releasing funds.
3. `removeDistributionConfig(Schema,address)` : Removes a distribution target if it is already allocated 0 distributions.

Recommendation: Consider documenting the risk and impact a compromised privileged role can cause on the protocol and inform the users in detail. As the privileged roles can be the single point of failure of the protocol, consider using a multi-sig or a contract with a timelock feature to mitigate the risk of being compromised or exploited.

VMC-22
Avoid Setting Large Shared Decimals on XVS Bridges

Informational | Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> The `sharedDecimals_` will be set to 8 on the bridges, as can be checked in the following deployment scripts:
>
> https://github.com/VenusProtocol/token-bridge/blob/develop/deploy/001-xvs-bridge-local.ts#L80
>
> https://github.com/VenusProtocol/token-bridge/blob/develop/deploy/002-xvs-bridge-remote.ts#L104

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The `BaseXVSProxyOFT` contract takes `sharedDecimals_` as a constructor parameter and sets the `ld2sdRate`. According to the code comments in `OFTCoreV2`,

> _sharedDecimals should be the minimum decimals on all chains

The XVS tokens will have 18 decimals on each EVM chain. However, it should be noted that there is an upper limit of `type(uint64).max` on `sd` amounts. This is required for bridging to non-EVM chains such as Aptos and Solana. If `sharedDecimals_` is set to 18, at most ~18 XVS tokens can be bridged in each call. As mentioned by the Layer Zero docs, "OFTV2 is intended to be used with no more than 10 shared decimals".

Recommendation: Consider deciding on suitable shared decimals for the XVS bridge so that the bridge amount in each call is not limited. Note that `sharedDecimals_` is declared an immutable variable, so it cannot be changed after the contract is deployed. If there are no plans to integrate with non-EVM chains, consider switching to OFTV1.

The Venus protocol team has confirmed that the `sharedDecimals_` will be set to 8 on the bridges. Therefore, this issue is marked as Informational.

VMC-23
XVS Transfer Amount from Whitelisted Users Is Considered in Daily Limit

• Informational Fixed

Update
Marked as "Fixed" by the Venus team. Addressed in: `a026422b599d32d1d1399631670e94804627e263`. The Venus team provided the following explanation:

> Change: we won't increase the consumption of the limits when the user is whitelisted

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The `_isEligibleToSend()` and `_isEligibleToReceive()` functions in the `BaseXVSProxyOFT` contract allow a whitelisted user to send or receive arbitrary token amount, which is not restricted by the single-transaction and daily limits. The bridges will still update the `chainIdToLast24HourTransferred` mapping to account for the transfer value in USD from whitelisted users.

As a result, large transfers from whitelisted users may block regular users' transfers within the same daily window if the whitelisted user's transaction gets executed first. On the other hand, regular users will not be affected if their transactions are executed before the whitelisted users. The execution order of transactions leads to different outcomes.

This issue is marked as Infomational as its validity and impact depend on the expected protocol behavior.

Recommendation: Confirm whether this is an intended protocol design, and if so, consider clarifying it in public-facing documents to let users be aware of such a scenario.

VMC-24 XVS Token's Name and Symbol Are Reversed

• Informational Fixed

Update
Marked as "Fixed" by the Venus team. Addressed in: `28bf1a85cddeb43769529433f929aae154eee177`.

File(s) affected: `isolated-pools/contracts/Bridge/token/XVS.sol`

Description: The `XVS` token inherits from the `ERC20` contract, initialized by `ERC20("XVS", "Venus XVS")`. The first parameter is the `name` of the ERC-20 token, while the second is the `symbol`. As a result, the `XVS` token will have the name `XVS` and the symbol of `Venus XVS`.

Recommendation: Consider using `ERC20("Venus XVS", "XVS")` instead. Also, note that on BSC, the name of the `XVS` token is `Venus` only.

VMC-25 Insufficient Validation in `isTrustedRemote()`

• Informational Fixed

Update
Since `XVSBridgeAdmin` is the owner of the bridge contracts, this issue is fixed on `XVSBridgeAdmin` by adding the `isTrustedRemote()` function and ensuring the `remoteAddress_` parameter is not `address(0)`.

Update
Marked as "Fixed" by the Venus team. Addressed in: `ed202ca6405b707c7687e4e1616b587578d1baa4`.

File(s) affected: `isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: The function `isTrustedRemote()` will return true if `_srcAddress` is an empty array and if the `_srcChainId` has not been added to `trustedRemoteLookup` yet.

Recommendation: Unfortunately, the function in question is not virtual so it cannot be overridden. Since there is another issue VMC-13 related to this contract, consider creating a new contract where the function `isTrustedRemote()` checks whether the input array is empty or not, similar to the function `getTrustedRemoteAddress()`.

VMC-26 Bypassable Events

• Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> Out of scope for now

File(s) affected: `governance-contracts/contracts/Governance/AccessControlManager.sol`

Description: In the function `AccessControlManager.giveCallPermission()`, the event `PermissionGranted` is bypassable by computing the `role` off-chain and calling the function `AccessControlManager.grantRole()` directly. Similarly, the event `PermissionRevoked` is bypassable by calling `AccessControlManager.revokeRole()` directly.

Recommendation: Consider how the aforementioned bypassable events could affect off-chain components that rely on emitted events.

VMC-27
Use of Solidity Version with Known Compiler Bugs

• Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> The bugs introduced in 0.8.13 shouldn't affect our codebase

File(s) affected: `isolated-pools/contracts/Bridge/`, `isolated-pools/contracts/BaseJumpRateModelV2.sol`, `isolated-pools/contracts/JumpRateModelV2.sol`, `isolated-pools/contracts/WhitePaperInterestRateModel.sol`, `isolated-pools/contracts/lib/constants.sol`, `oracle/contracts/ResilientOracle.sol`, `oracle/contracts/oracles/ChainlinkOracle.sol`, `venus-protocol/contracts/XVSVault/`, `protocol-reserve/contracts/ProtocolReserve/ProtocolShareReserve.sol`, `governance-contracts/contracts/Governance/AccessControlManager.sol`

Description: The above-listed contracts are compiled using Solidity version 0.5.16 or 0.8.13, which contains known compiler bugs, according to the Solidity official's compiler bug list. Specifically, Solidity version 0.8.13 includes one bug labeled as `medium/high` severity and two bugs labeled as `medium` severity. According to our examination, these bugs seem to not affect the code in scope. For 0.5.16, it is considered a relatively outdated version and is not recommended for production use.

Recommendation: Consider updating the Solidity version to the latest or a more recent version to avoid the code from potentially being affected by the compiler bugs. Note that the EVM version needs to be adjusted if a Solidity version >= 0.8.20 is used so that the bytecode does not include `PUSH0` opcodes.

VMC-28 Unlocked Pragma

• Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> It will be a breaking change because every contract in the core repo is using unlocked pragma

File(s) affected: `venus-protocol/contracts/XVSVault/XVSVaultErrorReporter.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.x.y`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

VMC-29
Address Aliasing May Affect Cross-Chain Access Control

Undetermined | Acknowledged

Update
Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> There will be separate permissions for timelocks to perform access-controlled actions

File(s) affected: `governance-contracts/contracts/Governance/AccessControlManager.sol`

Description: The protocol is planned to be deployed on Arbitrum and opBNB. Both chains allow users to send cross-chain messages to call L2 contracts from L1 so that the interaction with L2 protocols can continue even if the sequencer is down.

However, when an L2 contract receives a cross-chain call from L1, the `msg.sender` value is usually not the sender who initiates the cross-chain call on L1. On Arbitrum, the `msg.sender` is the aliased address of the original sender. On opBNB, which is built on the OP Stack, the `msg.sender` can be the L2 messenger contract or an aliased address, depending on which L1 contract is called and whether the original sender is a contract or not.

Therefore, access control mechanisms based on the `msg.sender` value should take such changes into account if a privileged role needs to call an L2 contract from L1 in certain circumstances, for example, in an emergency but directly sending L2 transaction is not possible since the sequencer is down.

In the past, the Uniswap Factory deployed on Arbitrum did not consider address aliasing, and its owner was set to the address of the Timelock contract on L1, causing the execution of governance proposals on Arbitrum to be temporarily blocked (reference).

Recommendation: If a privileged role may need to call the L2 contracts from L1, consider retrieving the original sender in the code so that the authorization does not fail. For implementation details, please refer to Arbitrum's `AddressAliasHelper` contract and Optimism's `CrossDomainOwnable3` contract.

VMC-30
Risks of Potential Operational Errors When Handling Failed Bridge Transfers

Low | Acknowledged

Update
This issue was identified during the fix review. Marked as "Acknowledged" by the Venus team. The Venus team provided the following explanation:

> The possible operational risk where when calling the `fallbackWithdraw()` function, we give an additional amount causing irregularities in the token supply across both chains is a good point. Since this function is permissioned and can only be executed via a VIP, we will take extra care as part of the VIP simulations and during the review period to ensure the value provided is amount is correct and does not create any issues.

File(s) affected: `token-bridge/contracts/Bridge/BaseXVSProxyOFT.sol`

Description: If a cross-chain transfer (from `XVSProxyOFTSrc` to `XVSProxyOFTDest`) fails on the destination chain, the `fallbackWithdraw()` function can be called on the source chain to reclaim the deposited tokens. However, there is no way of checking that there has ever been a failed transfer on a destination chain, and furthermore, there is no way of verifying the transfer amount. Therefore, it is possible to withdraw tokens on the source chain that were not ever "stuck" on a destination chain, which would prohibit users from bridging their tokens from a destination chain back to the source chain.

Exploit Scenario:

1. Assume 1,000 XVS tokens are minted on BSC. Thus, `XVSProxyOFTSrc` is also deployed on BSC, as it is the source chain for bridging. The `XVSProxyOFTDest` contract is deployed on Arbitrum and other L2s. Consider a situation where 900 XVS is given to a user, Alice, and 100 XVS is given to a user, Bob.
2. Alice bridges 900 XVS from BSC to Arbitrum. Bob also bridges all of his XVS tokens (100) to Arbitrum via the `XVSProxyOFTSrc` contract.
3. During the execution of bridging Bob's tokens, his 100 XVS tokens are transferred into the `XVSProxyOFTSrc` contract, and the LayerZero Endpoint contract is called to relay the cross-chain message.
4. On the destination chain (Arbitrum), the `XVSProxyOFTDest` contract is called by the LayerZero Endpoint contract, which mints 100 XVS to the destination address Bob submitted while initiating the token bridge.

5. Accidently, Bob entered the incorrect destination address, and the transfer of 100 XVS tokens failed due to the revert of the `callOnOFTReceived()` call. Since the transfer failed, the LayerZero message is stored in the `XVSProxyOFTDest` contract as a failed message, which can be retried at a later time.
6. The 100 XVS tokens are minted on Arbitrum but are kept in the `XVSProxyOFTDest` contract since the transfer failed.
7. Bob alerts the Venus team that his bridging transaction failed on the destination chain.
8. A variety of scenarios could occur:
    1. The Venus team calls `XVSProxyOFTDest.dropFailedMessage()` to remove Bob's failed bridge transaction on Arbitrum. The Venus team calls `XVSProxyOFTSrc.fallbackWithdraw()` for 100 XVS tokens, which claims the 100 XVS that Bob initially bridged. Then, the 100 XVS tokens are returned to Bob. Note that there are still 100 XVS tokens minted on Arbitrum that are now trapped in the `XVSProxyOFTDest` contract.
    2. The Venus team erroneously calls `XVSProxyOFTSrc.fallbackWithdraw()` for 200 XVS tokens (rather than the correct amount of 100 XVS), which claims 100 XVS that Bob initially bridged as well as an additional 100 XVS tokens. Now, if Alice wants to bridge her 900 XVS from Arbitrum to BSC, the bridge will fail since there is an insufficient XVS balance in the `XVSProxyOFTSrc` contract. Therefore, Alice would only be able to bridge 800 of her 900 XVS tokens back to BSC.

Recommendation: One solution would be to refactor the function `XVSProxyOFTDest.dropFailedMessage()` so that when called, the failed message is deleted, and the tokens from the failed transfer are bridged back to the source chain. This would also solve the problem of unclaimable minted tokens on the destination chain by burning the tokens during the bridging back to the source chain. Then, when the tokens are returned to the source chain, they are either:

1. Transferred back to the original sender of the bridged tokens.
2. The tokens are kept in the source contract, where the amount of returned tokens is tracked as a global state. Then, the Venus team can withdraw returned tokens up to the amount tracked.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Acknowledged For the `VTreasuryV8.withdrawTreasuryToken()` and `withdrawTreasuryNative()` functions, the comment for the parameter `withdrawAmount` can be changed since the address receiving the funds is not necessarily the owner.
2. Acknowledged There are typos in the code comments of the `TokenController.minterToCap` and `minterToMintedAmount` variables.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `231...9eb ./isolated-pools/contracts/JumpRateModelV2.sol`
- `f57...3fd ./protocol-reserve/contracts/ProtocolReserve/ProtocolShareReserve.sol`
- `5cd...a3b ./isolated-pools/contracts/WhitePaperInterestRateModel.sol`
- `a34...bff ./venus-protocol/contracts/Governance/VTreasuryV8.sol`
- `af6...9fb ./isolated-pools/contracts/lib/constants.sol`

- `766...f61 ./oracle/contracts/oracles/ChainlinkOracle.sol`
- `b3f...7d3 ./governance-contracts/contracts/Governance/AccessControlManager.sol`
- `42c...a8e ./oracle/contracts/ResilientOracle.sol`
- `6ee...df8 ./isolated-pools/contracts/BaseJumpRateModelV2.sol`
- `c6f...70b ./venus-protocol/contracts/XVSVault/XVSVaultStorage.sol`
- `31e...891 ./venus-protocol/contracts/XVSVault/XVSStore.sol`
- `e8f...495 ./venus-protocol/contracts/XVSVault/XVSVaultProxy.sol`
- `f0b...fd3 ./venus-protocol/contracts/XVSVault/XVSVault.sol`
- `d56...eaa ./venus-protocol/contracts/XVSVault/XVSVaultErrorReporter.sol`
- `cbe...5db ./isolated-pools/contracts/Bridge/BaseXVSProxyOFT.sol`
- `50f...95c ./isolated-pools/contracts/Bridge/XVSBridgeAdmin.sol`
- `b8a...6e1 ./isolated-pools/contracts/Bridge/XVSProxyOFTDest.sol`
- `881...888 ./isolated-pools/contracts/Bridge/XVSProxyOFTSrc.sol`
- `fb3...33d ./isolated-pools/contracts/Bridge/interfaces/IXVSProxyOFT.sol`
- `783...515 ./isolated-pools/contracts/Bridge/interfaces/IXVS.sol`
- `e5d...bbe ./isolated-pools/contracts/Bridge/token/XVS.sol`
- `8e1...070 ./isolated-pools/contracts/Bridge/token/TokenController.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Non-false positive findings have been included in the report.

Test Suite Results

The test suites were run by calling `npx hardhat test`. Only the relevant tests have been included in the output. All tests passed.

1. Venus Treasury V8

```
No tests are provided for the in-scope files.
```

2. XVS Bridge

```
Bridge Admin:
    ✔ Revert if EOA called owner function of bridge
    ✔ Revert if permisssions are not granted to call owner functions of bridge (81ms)
    ✔ Success if permisssions are granted to call owner functions of bridge (875ms)
    ✔ Revert if function is removed from function registry
    ✔ Revert if function is not found in bridge admin function registry
    ✔ Success on trnafer bridge owner

  Proxy OFTV2:
    ✔ send tokens from proxy oft and receive them back (206ms)
    ✔ Reverts if single transaction limit exceed (43ms)
    ✔ Reverts if single transaction limit exceed on remote chain (203ms)
```

```
    ✔ Reverts if max daily transaction limit exceed (636ms)
    ✔ Reverts if max daily transaction limit exceed on remote chain (745ms)
    ✔ Reset limit if 24hour window passed (900ms)
    ✔ Reverts on remote chain if minting permission is not granted to remoteOFT (95ms)
    ✔ Reverts on remote chain if minting cap is reached (162ms)
    ✔ Reverts initialy and should success on retry (130ms)
    ✔ Reverts on remote chain if bridge is paused (203ms)
    ✔ Reverts on remote chain if xvs token is paused (90ms)
    ✔ Reverts if amount is too small (49ms)
```

3. Isolated Pool Changes

```
  Jump rate model tests
    ✔ Update jump rate model (149ms)
    ✔ Utilization rate: borrows and badDebt is zero
    ✔ Utilization rate
    ✔ Borrow Rate: below kink utilization (56ms)
    ✔ Borrow Rate: above kink utilization (70ms)
    ✔ Supply Rate

  White paper interest rate model tests
    ✔ Model getters (41ms)
    ✔ Utilization rate: borrows and badDebt is zero
    ✔ Utilization rate
    ✔ Borrow Rate (83ms)
    ✔ Supply Rate (46ms)

  VToken
    setProtocolSeizeShare
      ✔ reverts if access control manager does not allow the call (139ms)
      ✔ reverts if the provided seize share is larger than the liquidation incentive minus one (150ms)
      ✔ updates protocolSeizeShare and emits an event on success (151ms)
    set access control manager
      ✔ reverts if not an owner set access control manager (64ms)
      ✔ success by admin (64ms)
    set interestRateModel
      ✔ reverts if rejected by access control manager (84ms)
      ✔ success if allowed to set interest rate model (103ms)
    set reserve factor
      ✔ reverts if rejected by access control manager (86ms)
      ✔ success if allowed to set setReserveFactor (91ms)
    setProtocolShareReserve
      ✔ reverts if called by a non-owner (70ms)
      ✔ reverts if zero address (67ms)
      ✔ sets protocol share reserve if called by admin (69ms)
    setShortfallContract
      ✔ reverts if called by a non-owner (68ms)
      ✔ reverts if zero address (69ms)
      ✔ sets shortfall contract if called by admin (70ms)
```

4. Resilient Oracles Changes

```
  Binance Oracle unit tests
    ✔ set price
    ✔ set BNB price
    ✔ fetch price (71ms)
    ✔ fetch BNB price
    ✔ price expired (96ms)
    ✔ set WBETH price
    ✔ fetch WBETH price

  bound validator
    add validation config
      ✔ length check
      ✔ validation config check (42ms)
      ✔ config added successfully & event check
    validate price
      ✔ validate price (97ms)
```

```
Oracle unit tests
  set token config
    ✔ cannot set feed to zero address
    ✔ sets a token config
  batch set token configs
    ✔ cannot set feed or vtoken to zero address
    ✔ parameter length check
    ✔ set multiple feeds
  getPrice
    ✔ gets the price from Chainlink for vBNB
    ✔ gets the price from Chainlink for USDC
    ✔ gets the price from Chainlink for USDT
    ✔ gets the price from Chainlink for DAI
    ✔ gets the direct price of a set asset
    ✔ reverts if no price or feed has been set
  setDirectPrice
    ✔ sets the direct price
  stale price validation
    ✔ stale price period cannot be 0
    ✔ modify stale price period will emit an event
    ✔ revert when price stale (59ms)
    ✔ if updatedAt is some time in the future, revert it
    ✔ the chainlink anwser is 0, revert it (88ms)

Oracle plugin frame unit tests
  admin check
    ✔ transfer owner (46ms)
  token config
    add single token config
      ✔ token can"t be zero & maxStalePeriod can't be zero
      ✔ token config added successfully & events check
    batch add token configs
      ✔ length check
      ✔ token config added successfully & data check
  get underlying price
    ✔ revert when asset not exist
    ✔ revert when price is expired
    ✔ revert when price is not positive (just in case Pyth return insane data) (88ms)
    ✔ price should be 18 decimals (80ms)
  validation
    ✔ validate price (138ms)
    ✔ validate BNB price (151ms)

Twap Oracle unit tests
  token config
    add single token config
      ✔ should revert on calling updateTwap without setting token configs
      ✔ vToken can"t be zero & pool address can't be zero & anchorPeriod can't be 0 (67ms)
      ✔ reset token config (87ms)
      ✔ token config added successfully & events check
    batch add token configs
      ✔ length check
      ✔ token config added successfully & data check
  update twap
    ✔ revert if get underlying price of not existing token
    ✔ revert if get underlying price of token has not been updated
    ✔ twap update after multiple observations (57ms)
    ✔ should delete observation which does not fall in current window and add latest observation (63ms)
    ✔ should pick last available observation if none observations are in window and also delete
privious one (77ms)
    ✔ should add latest observation after delete observations which does not fall in current window
(64ms)
    ✔ should delete multiple observation and pick observation which falling under window (151ms)
    ✔ cumulative value (129ms)
    ✔ test reversed pair (68ms)
    ✔ twap calculation for non BNB based token (151ms)
    twap calculation for BNB based token
      ✔ if no BNB config is added, revert
      ✔ twap calculation (215ms)
  validation
    ✔ validate price (137ms)
```

```
Oracle plugin frame unit tests
  token config
    add single token config
      ✔ vToken can"t be zero & main oracle can't be zero
      ✔ reset token config (122ms)
      ✔ token config added successfully & events check (69ms)
    batch add token configs
      ✔ length check
      ✔ token config added successfully & data check (167ms)
  change oracle
    set oracle
      ✔ null check (110ms)
      ✔ existance check (45ms)
      ✔ oracle set successfully & data check (100ms)
  get underlying price
    ✔ revert when protocol paused (49ms)
    ✔ revert price when main oracle is disabled and there is no fallback oracle
    ✔ revert price main oracle returns 0 and there is no fallback oracle
    ✔ revert if price fails checking
    ✔ check price with/without pivot oracle (66ms)
    ✔ disable pivot oracle (51ms)
    ✔ enable fallback oracle (145ms)
    ✔ Return fallback price when fallback price is validated successfully with pivot oracle (43ms)
    ✔ Return main price when fallback price validation failed with pivot oracle
```

5. XVS Vault

```
XVSVault
  setXvsStore
    ✔ fails if XVS is a zero address
    ✔ fails if XVSStore is a zero address
    ✔ fails if the vault is already initialized
  add
    ✔ reverts if ACM does not allow the call (53ms)
    ✔ reverts if xvsStore is not set (57ms)
    ✔ reverts if a pool with this (staked token, reward token) combination already exists (88ms)
    ✔ reverts if staked token exists in another pool (68ms)
    ✔ reverts if reward token is a zero address (55ms)
    ✔ reverts if staked token is a zero address (54ms)
    ✔ reverts if alloc points parameter is zero (57ms)
    ✔ emits PoolAdded event (98ms)
    ✔ adds a second pool to an existing rewardToken (159ms)
    ✔ sets pool info (121ms)
    ✔ configures reward token in XVSStore (119ms)
  set
    ✔ reverts if ACM does not allow the call (57ms)
    ✔ reverts if pool is not found (58ms)
    ✔ reverts if total alloc points after the call is zero (85ms)
    ✔ succeeds if the pool alloc points is zero but total alloc points is nonzero (380ms)
    ✔ emits PoolUpdated event (92ms)
setRewardAmountPerBlock
  ✔ reverts if ACM does not allow the call (55ms)
  ✔ reverts if the token is not configured in XVSStore (113ms)
  ✔ emits RewardAmountPerBlockUpdated event (112ms)
  ✔ updates reward amount per block (134ms)
setWithdrawalLockingPeriod
  ✔ reverts if ACM does not allow the call (56ms)
  ✔ reverts if pool does not exist (55ms)
  ✔ reverts if the lock period is 0 (56ms)
  ✔ reverts if the lock period is absurdly high (56ms)
  ✔ emits WithdrawalLockingPeriodUpdated event (124ms)
  ✔ updates lock period (202ms)
pendingReward
  ✔ includes the old withdrawal requests in the rewards computation (367ms)
  ✔ excludes the new withdrawal requests from the rewards computation (504ms)
deposit
  ✔ reverts if the vault is paused (89ms)
  ✔ reverts if pool does not exist
  ✔ transfers pool token to the vault (159ms)
```

```
    ✔ updates user's balance (162ms)
    ✔ fails if there's a pre-upgrade withdrawal request (237ms)
    ✔ succeeds if the pre-upgrade withdrawal request has been executed (771ms)
    ✔ uses the safe _transferReward under the hood (451ms)
  executeWithdrawal
    ✔ fails if the vault is paused (87ms)
    ✔ only transfers the requested amount for post-upgrade requests (471ms)
    ✔ handles pre-upgrade withdrawal requests (473ms)
    ✔ handles pre-upgrade and post-upgrade withdrawal requests (737ms)
  requestWithdrawal
    ✔ fails if the vault is paused (90ms)
    ✔ transfers rewards to the user (399ms)
    ✔ uses the safe _transferReward under the hood (416ms)
    ✔ fails if there's a pre-upgrade withdrawal request (211ms)
  claim
    ✔ fails if there's a pre-upgrade withdrawal request (109ms)
    ✔ succeeds if the pre-upgrade withdrawal request has been executed (475ms)
    ✔ excludes pending withdrawals from the user's shares (544ms)
    ✔ correctly accounts for updates in reward per block (348ms)
    ✔ uses the safe _transferReward under the hood (228ms)
  _transferReward
    ✔ sends the available funds to the user (181ms)
    ✔ emits VaultDebtUpdated event if vault debt is updated (132ms)
    ✔ does not emit VaultDebtUpdated event if vault debt is not updated (158ms)
    ✔ records the pending transfer (158ms)
    ✔ records several pending transfers (321ms)
    ✔ sends out the pending transfers in addition to reward if full amount <= funds available (540ms)
    ✔ sends a part of the pending transfers and reward if full amount > funds available (501ms)
  pendingWithdrawalsBeforeUpgrade
    ✔ returns zero if there were no pending withdrawals
    ✔ returns zero if there is only a new-style pending withdrawal (212ms)
    ✔ returns the requested amount if there is an old-style pending withdrawal (73ms)
    ✔ returns the total requested amount if there are multiple old-style pending withdrawals (119ms)
    ✔ returns zero if the pending withdrawal was executed (250ms)
  Scenarios
    ✔ works correctly with multiple claim, deposit, and withdrawal requests (1680ms)
```

6. Protocol Share Reserves

```
  ProtocolShareReserve: Tests
    ✔ check configuration of schemas (58ms)
    ✔ update configuration of schemas (99ms)
    ✔ remove configuration (246ms)
    ✔ collect and distribute of income (617ms)
```

7. Access Control Manager

```
  Access Control
    Access Control
      ✔ only default admin role can give call permissions (159ms)
      ✔ should not have permissions (47ms)
      ✔ should have permissions (63ms)
      ✔ should revoke role (74ms)
      ✔ should be able to call the function only for the given contract (41ms)
      ✔ should be able to call the function on every contract  (38ms)
```

Code Coverage

Coverage was gathered by running `npx hardhat coverage`. Only the files in scope are included in the output.

1. Venus Treasury V8

```
No tests are provided for the in-scope files.
```

2. XVS Bridge

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/Bridge/	94.12	61.11	83.33	93.58	
BaseXVSProxyOFT.sol	91.49	58.7	73.33	90.91	... 198,199,215
XVSBridgeAdmin.sol	100	75	85.71	100	
XVSProxyOFTDest.sol	100	66.67	100	100	
XVSProxyOFTSrc.sol	88.89	50	100	90.91	60
contracts/Bridge/interfaces/	100	100	100	100	
IXVS.sol	100	100	100	100	
IXVSProxyOFT.sol	100	100	100	100	
contracts/Bridge/token/	76.67	66.67	69.23	73.68	
TokenController.sol	70.83	62.5	60	68.75	... 132,141,152
XVS.sol	100	75	100	100	

3. Isolated Pool Changes

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.6	76.28	95.51	95.75	
BaseJumpRateModelV2.sol	100	70	100	94.12	94,146
JumpRateModelV2.sol	100	100	100	100	
WhitePaperInterestRateModel.sol	100	75	100	94.12	101
contracts/lib/	100	88.89	100	100	
constants.sol	100	100	100	100	

4. Resilient Oracles Changes

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	83.33	79.63	85	84.81	
ResilientOracle.sol	83.33	79.63	85	84.81	... 404,433,449
contracts/oracles/	92.89	82.84	97.56	96.74	
ChainlinkOracle.sol	100	92.31	100	100	

5. XVS Vault

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
Vault/	49.49	47.5	51.85	50.49	
XVSVault/	64.31	52.21	59.09	64.9	
XVSStore.sol	57.69	50	66.67	60.71	... 70,72,73,89
XVSVault.sol	72.85	56.25	67.35	73.47	... 822,828,829
XVSVaultErrorReporter.sol	0	100	0	0	26,28,35,37
XVSVaultProxy.sol	0	0	0	0	... 120,130,132
XVSVaultStorage.sol	100	100	100	100	

6. Protocol Share Reserves

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
ProtocolReserve/	84.76	58.33	88.89	86.08	
ProtocolShareReserve.sol	84.76	58.33	88.89	86.08	... 542,543,545

7. Access Control Manager

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
Governance/	59.46	43.98	45.1	54.95	
AccessControlManager.sol	100	100	100	100	

Changelog

- 2023-10-27 - Initial report
- 2023-12-19 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp’s mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp’s team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp’s collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp