




Venus Protocol (Vaults)

Executive Summary

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

Type	Staking & Vaults	Documentation quality	Low	<div><div></div></div>
Timeline	2023-05-08 through 2023-05-19	Test quality	Low	<div><div></div></div>
Language	Solidity	Total Findings	25	<div><div></div></div> <div>Unresolved: 2Fixed: 11</div> <div>Acknowledged: 7Mitigated: 5</div>
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review	High severity findings ⓘ	3	<div><div></div></div> <div>Fixed: 3</div>
Specification	Venus Docs 	Medium severity findings ⓘ	4	<div><div></div></div> <div>Fixed: 4</div>
Source Code	<ul style="list-style-type: none">VenusProtocol/venus-protocol #f1426f0 	Low severity findings ⓘ	13	<div><div></div></div> <div>Unresolved: 2Fixed: 3</div> <div>Acknowledged: 4Mitigated: 4</div>
Auditors	<ul style="list-style-type: none">Ibrahim Abouzied Auditing EngineerJulio Aguliar Auditing EngineerRoman Rohleder Senior Auditing EngineerShih-Hung Wang Auditing Engineer	Undetermined severity findings ⓘ	2	<div><div></div></div> <div>Fixed: 1Mitigated: 1</div>
		Informational findings ⓘ	3	<div><div></div></div> <div>Acknowledged: 3</div>

Summary of Findings

The Venus Protocol is a money market on the Binance Smart Chain. This audit was focused on an upgrade to the VAI, VRT, and XVS Vaults. Users depositing the VAI stable coin into the VAI Vault are rewarded with XVS tokens. The VRT Vault, which allows users to earn interest on the VRT legacy token, is being updated to stop accruing interest as part of its deprecation. The XVS vault is a generalized vault that supports staking in different token pools. It also tracks users voting power through their number of staked XVS tokens.

During the audit, we focused on identifying the impacts of the contract upgrades and whether they could lead to locked funds or new attack vectors. Though we did not find any major issues relating to upgradeability, we did uncover findings in the existing contract logic. The accounting in XVS Vault will break if pools share the same pool token (**VENUS-1**). There also exist conditions in which voting power can be manipulated (**VENUS-2** & **VENUS-3**). We also detail smaller accounting discrepancies and access control improvements in the report.

The Venus team was able to quickly address all of our questions during the audit. The test suite has room for improvement. The line coverage for the vault contracts is between 57-71%. We recommend getting these values as close to 100% as possible. Documentation was inaccessible at times, with some links on the website not working.

We recommend addressing all of the listed issues to ensure the sustainability and longevity of the protocol.

Update: The Venus Protocol team has addressed the majority of the issues. They have chosen to leave some of the findings unaddressed and out of scope for this current release. We encourage users to read these findings, most notably **VENUS-9**, before interacting with the protocol.

ID	DESCRIPTION	SEVERITY	STATUS
VENUS-1	XVS Vault Users Receive Fewer Rewards if Multiple Pools Have the Same Pool Token	• High ⓘ	Fixed
VENUS-2	Use of Unsafe Cast Operations Leading to Incorrect Voting Power	• High ⓘ	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
VENUS-3	Stakers in XVS Vault May Be Able to Double Their Voting Power by Re-Staking	<ul style="list-style-type: none"> High ⓘ 	Fixed
VENUS-4	VRT Vault Users Can Earn Additional Interests When Depositing After the Last Accrual Block	<ul style="list-style-type: none"> Medium ⓘ 	Fixed
VENUS-5	VAIVault Is Susceptible to MEV Bots	<ul style="list-style-type: none"> Medium ⓘ 	Fixed
VENUS-6	Initializers Can Be Called More than Once	<ul style="list-style-type: none"> Medium ⓘ 	Fixed
VENUS-7	Re-Entrancy Guards Bypassable by Admins in XSVVault.sol and VAIVault.sol	<ul style="list-style-type: none"> Medium ⓘ 	Fixed
VENUS-8	Critical Role Transfer Not Following Two-Step Pattern	<ul style="list-style-type: none"> Low ⓘ 	Fixed
VENUS-9	Inaccurate Pending Rewards for VAI Vault Users	<ul style="list-style-type: none"> Low ⓘ 	Unresolved
VENUS-10	VRT Vault Interests Are Accrued During the Pausing Period	<ul style="list-style-type: none"> Low ⓘ 	Acknowledged
VENUS-11	Inaccurate Calculation of Pending Rewards of XVS Vault Pools	<ul style="list-style-type: none"> Low ⓘ 	Fixed
VENUS-12	Inaccurate Calculation of Stake Amount for XVS Vault Pools	<ul style="list-style-type: none"> Low ⓘ 	Mitigated
VENUS-13	Vaults May Not Deliver Full Payouts	<ul style="list-style-type: none"> Low ⓘ 	Mitigated
VENUS-14	Considerations on Claiming Rewards for Other Users for the VAI and XVS Vaults	<ul style="list-style-type: none"> Low ⓘ 	Mitigated
VENUS-15	A Compromised Admin Address Can Drain All of VRT	<ul style="list-style-type: none"> Low ⓘ 	Acknowledged
VENUS-16	Privileged Roles and Ownership	<ul style="list-style-type: none"> Low ⓘ 	Acknowledged
VENUS-17	XVS Vault's Domain Separator Does Not Include a Contract Version	<ul style="list-style-type: none"> Low ⓘ 	Unresolved
VENUS-18	Missing Input Validation	<ul style="list-style-type: none"> Low ⓘ 	Mitigated
VENUS-19	Use of Outdated Compiler Version	<ul style="list-style-type: none"> Low ⓘ 	Acknowledged
VENUS-20	Pool May Be Permanently Misconfigured	<ul style="list-style-type: none"> Low ⓘ 	Fixed
VENUS-21	VAI Losing 1\$ Peg May Lead to a Drained VAIVault and Destabilize the XVS Token and Vault	<ul style="list-style-type: none"> Informational ⓘ 	Acknowledged
VENUS-22	Application Monitoring Can Be Improved by Emitting More Events	<ul style="list-style-type: none"> Informational ⓘ 	Acknowledged
VENUS-23	Unlocked Pragma	<ul style="list-style-type: none"> Informational ⓘ 	Acknowledged
VENUS-24	Vaults May Be Insufficiently Funded	<ul style="list-style-type: none"> Undetermined ⓘ 	Mitigated
VENUS-25	VRTVault May Be Operated and Used Past the Last Accruing Block	<ul style="list-style-type: none"> Undetermined ⓘ 	Fixed

Assessment Breakdown

i 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

- contracts/VRTVault/VRTVault.sol
- contracts/VRTVault/VRTVaultStorage.sol
- contracts/Vault/VAIVault.sol
- contracts/Vault/VAIVaultStorage.sol
- contracts/XVSVault/XVSSStore.sol
- contracts/XVSVault/XVSVault.sol
- contracts/XVSVault/XVSVaultStorage.sol

Findings

VENUS-1

XVS Vault Users Receive Fewer Rewards if Multiple Pools Have the Same Pool Token

• **High** ⓘ **Fixed**

i Update

Marked as "Mitigated" by the client. Addressed in: `4404d27c57d40bea97320a86437ba662e26b85ec` . The client provided the following explanation:

We prohibited pools with the same staked tokens

File(s) affected: `XVSVault.sol`

Description: When updating the rewards for the pools in the XVS vault, `pool.token.balanceOf(address(this))` is used as a pool's `supply` (i.e., the total deposits in the pool). However, if multiple pools have the same type of pool token, using `balanceOf()` returns the total deposits across all the pools instead of a specific one. As a result, the value of `supply` will be overestimated, causing pool users to receive fewer rewards than they should.

Exploit Scenario:

- Starting from an initially empty state, the admin adds the following two pools:
 - Pool A
 - Reward token: XVS
 - Allocation point: 100
 - Pool token: XVS
 - Reward per block: 100
 - Lock period: Any valid value
 - Pool B
 - Reward token: USDC
 - Allocation point: 100
 - Pool token: XVS
 - Reward per block: 100
 - Lock period: Any valid value
- Suppose the XVS store holds enough XVS and USDC tokens as rewards. Alice deposits into pool A with 100 XVS, and Bob deposits into pool B with 100 XVS.
- One block has passed. Since 100 XVS is rewarded for each block, Pool A is the only pool that rewards XVS, and Alice is the only staker, she should get the entire XVS 100 reward. Similarly, Bob should get 100 USDC as the reward.
- However, according to the implementation, Alice can only get 50 XVS as the reward, and Bob can only get 50 USDC. The remaining 50 XVS and USDC rewards are left in the XVS store and not distributed to the stakers.

Recommendation: Consider using a state variable to keep track of the deposit amount in each pool separately instead of using `pool.token.balanceOf()` as the deposited amount.

VENUS-2

Use of Unsafe Cast Operations Leading to Incorrect Voting Power

• High ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. Addressed in: `3fe0317a06936b1992929e92727c6b005a9fdffc`. The client provided the following explanation:

Fixed in 3fe0317a. Note, however, that amounts of XVS (which is used as governance token) can't possibly be larger than `type(uint96).max`.

File(s) affected: `XVSVault.sol`

Description: The following functions make use of unsafe cast operations (`uint96()`), exposing them to truncation, when reaching values higher than `type(uint96).max` (`79,228,162,514,264,337,593,543,950,335`):

- `deposit()`: Depositor could end up with less votes than he is entitled to.
- `requestWithdrawal()`: Withdrawer is debited a too small amount.
- `getStakeAmount()`: May return a truncated (too small) staked amount.

Exploit Scenario:

- Attacker deposits more than `type(uint96).max` in funds using `deposit()` (at most in chunks of `type(uint96).max`, to circumvent issue 1.).
- Attacker requests a withdrawal of more than `type(uint96).max` in funds using `requestWithdrawal()`.
- Attacker waits out the corresponding lockup period for the given token pool.
- Attacker executes the requested withdrawal using `executeWithdrawal()`.
- Attacker ends up back with all his deposited funds, accumulated rewards in reward token and a surplus in votes, that have wrongfully not been deducted due to issue 2.

Recommendation: We recommend the use of a [safe cast library](#), or the already present function `safe96()`.

VENUS-3

Stakers in XVS Vault May Be Able to Double Their Voting Power by Re-Staking

• High ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. Addressed in: `bad686c0ebd7c24e6f13cd7115a521c2a436bed2`.

File(s) affected: XVSVault.sol

Description: Users can deposit XVS into the XVS vault to get voting power. However, according to the implementation, when users request a withdrawal, their voting power is removed only when the pool's reward token is XVS. For pools having a different reward token, the user's voting power will not be removed, and therefore they can re-stake to the pool to gain double voting power with the same amount of XVS deposit.

Exploit Scenario:

1. Suppose there is a pool with USDC as the reward token and XVS as the pool token.
2. Assume that Alice has not made any deposits yet. First, Alice delegates to herself.
3. Alice deposits 100 XVS to the pool. Since the pool token is XVS, Alice gets 100 voting power.
4. Alice sends a withdrawal request. Since the reward token is not XVS, Alice's voting power is not removed.
5. After the locked period passes, Alice can execute the withdrawal and re-stake into an XVS pool to get double voting power.

Recommendation: Consider changing the logic in `requestWithdrawal()` so that the user's voting power is removed when the pool token is XVS, rather than the reward token being XVS.

VENUS-4

VRT Vault Users Can Earn Additional Interests When Depositing After the Last Accrual Block

• Medium ⓘ Fixed

✓ **Update**

Marked as "Fixed" by the client. Addressed in: `b0a896c88bef287c6a58e1fe51aa63e8cb4f2dab`. The client provided the following explanation:

Fixed by putting more restrictions in `setLastAccruingBlock` (see #25). Now admin cannot update `lastAccruingBlock` after it has passed.

File(s) affected: VRTVault.sol

Description: When a user deposits into the VRT vault and `lastAccruingBlock` is behind the current block, their `accrualStartBlockNumber` will be set to `lastAccruingBlock`. Although the user deposits at the current block, their interests are accrued starting from the `lastAccruingBlock`. They can claim those interests once the `lastAccruingBlock` has been moved forward to a more recent block.

Exploit Scenario:

1. Suppose the `lastAccruingBlock` is set to 100 and the current block number is 200.
2. A new user deposits into the vault. According to the `deposit()` function, their `accrualStartBlockNumber` is set to 100, the same value as `lastAccruingBlock`.
3. The admin updates the `lastAccruingBlock` to 150.
4. The user calls `claim()`. According to the calculation in `computeAccruedInterest()`, the user receives the interests between block 100 and 150 even though they did deposit any funds during that period.

This issue also exists for existing users with a non-zero deposit at the time of the second deposit. As a result, they will earn more interest for their second deposit when the `lastAccruingBlock` moves forward.

Recommendation: It is unclear whether the protocol should allow users to deposit after the last accruing block. If so, consider re-designing how the VRT vault keeps track of users' `accrualStartBlockNumber`. Ideally, each deposit should have its specific `accrualStartBlockNumber`.

An alternative way is to allow users to deposit only when their deposit amount is 0 or `lastAccruingBlock` is at least the current block. However, this approach limits the flexibility of the deposit functionality.

VENUS-5 VAIVault Is Susceptible to MEV Bots

• Medium ⓘ Fixed

✓ **Update**

Marked as "Fixed" by the client. Addressed in: `49db8b4151768ec3fb76c429386f0e544572ee03`.

File(s) affected: VAIVault.sol

Description: After the XVS rewards are transferred to the VAI vault, someone has to call `updatePendingRewards()` to make the vault update the internal accounting for pending rewards. Typically, when a user operates (e.g., borrow, redeem) on a vToken, the `Comptroller` transfers the XVS tokens to the VAI vault and calls the update function in the same transaction.

An attacker can exploit this design to get most of the rewards by depositing a large number of funds, triggering the reward emission, and then withdrawing their deposit. As a result, they do not have to stake VAI in the vault for some period but can still get rewards from it.

The attacker might not be incentivized to launch such an attack if the rewards are relatively small. However, this could be an issue if the accumulated XVS rewards are large enough (e.g., a long period has passed, but no one operates on any vToken).

In addition to the MEV exploit, the current implementation could lead to improper accounting. After someone sends the XVS reward to the VAI vault, the `updatePendingRewards()` function has to be executed before any other user operations such as `deposit()` or `withdraw()`. Otherwise, users can lose their eligible rewards, which can be locked in the vault permanently.

At the beginning of every user operation, the `updateAndPayOutPending()` function is called, which claims the rewards for the user and calls the `safeXVSTransfer()` function. The bug occurs when the `safeXVSTransfer()` function updates the `xvsBalance` state variable according to the current XVS balance. Since the current XVS balance may include pending rewards that are not accounted for, `xvsBalance` should not be overwritten with the current XVS balance.

- Exploit Scenario:**
1. Suppose the total deposit in the VAI vault is 1M, and the accumulated XVS rewards for the vault are 100.
 2. Before anyone else triggers the XVS reward distribution, the attacker does the following Steps 3. to 6. in a single transaction:
 3. The attacker flash loans 9M VAI and deposits it into the vault.
 4. They trigger the `Comptroller` to send the pending 100 XVS rewards to the vault and call `VAIVault.updatePendingRewards()` to make the vault account for the rewards.
 5. The attacker is now eligible for the rewards since they deposited before the reward emission. Then, they withdraw their entire deposit. According to the reward calculation, they can get 90% of the rewards.
 6. The attacker repays the flash loan. As a result, they get 90 XVS without actually staking in the vault for some period.

Recommendation: Consider calling `updatePendingRewards()` at the beginning of the `updateVault()` function. This way, all pending rewards will be accounted for before the attacker deposits, making the attack unprofitable.

VENUS-6 Initializers Can Be Called More than Once

• Medium ⓘ Fixed

✓ Update

Marked as "Fixed" by the client. Addressed in: `c02ccb8e0bad620ede890c9bf411f98d2c2e8e75` , `53d11560f8a836b999b76405f3842757ebb6fb82` .

File(s) affected: `VAIVault.sol` , `XVSVault.sol`

Description: `VAIVault.setVenusInfo()` and `XVSVault.setXvsStore()` functions are used to set the different token addresses. However, the admin can call the function again and change the addresses, thereby locking the tokens in the vault.

Recommendation: Do not allow calls to `VAIVault.setVenusInfo()` and `XVSVault.setXvsStore()` once the contract has been initialized.

VENUS-7 Re-Entrancy Guards Bypassable by Admins in `XVSVault.sol` and `VAIVault.sol`

• Medium ⓘ Fixed

✓ Update

Marked as "Fixed" by the client. Addressed in: `c02ccb8e0bad620ede890c9bf411f98d2c2e8e75` , `53d11560f8a836b999b76405f3842757ebb6fb82` .

File(s) affected: `XVSVault.sol` , `VAIVault.sol`

Description: All three vault contracts `XVSVault.sol` , `VRTVault.sol` and `VAIVault.sol` use a custom re-entrancy guard modifier (`nonReentrant`) (i.e. rather than relying on the [well-established one from OpenZeppelin](#)).

While otherwise it works as expected, it is problematic in `XVSVault.sol` , where it is initialized in function `setXvsStore()` , which however can be called multiple times.

The same issue applies to `VAIVault.setVenusInfo()` .

- Exploit Scenario:** This could be abused by a malicious or otherwise compromised `admin` (after initialization):
1. `admin` calls a `nonReentrant` modifier protected function (i.e. `deposit()`).
 2. He is able to trigger a re-entrancy due to one of the transfer functions back into his attacking contract.
 3. From there he calls `setXvsStore(PREVIOUS_XVS_ADDR, PREVIOUS_XVS_STORE_ADDR)` , effectively not changing the XVS/XVS store addresses, however disabling the re-entrancy guard, due to `L778 : _notEntered = true;`
 4. He repeats steps 1. to 3. until achieving his goal.

Recommendation: Consider restricting the admin to calling the `XVSVault.setXvsStore()` and `VAIVault.setVenusInfo()` functions only once.

VENUS-8 Critical Role Transfer Not Following Two-Step Pattern

• Low ⓘ

Fixed

✓ Update

This issue was resolved by removing the ability to transfer roles entirely.

Marked as "Fixed" by the client. Addressed in: `c2779f0c87b9da7bf7f62f09026f93a3e49c78a7` .

File(s) affected: `VAIVault.sol`

Description: The `admin` role can be transferred to another address simply by calling the `VAIVault.setNewAdmin()` function. This function immediately transfers a high-level privilege to new addresses in a single transaction, which can be risky from a security perspective, as providing a faulty address may lock out that role from future calls.

A more secure pattern for such privilege transfers is to require the new pending addresses to issue an `acceptAdmin()` function call before finalizing the transfer. Note that this pattern is common even with the use of timelocks, such as the [Compound Timelock](#) contract.

Recommendation: Make sure that before transferring an authority the new account makes a call to the `acceptAdmin()` method to accept the role. Consider using the [OpenZeppelin's Ownable2Step library](#) (`Ownable2StepUpgradeable`) as reference or equivalent functions as used in the other vaults.

VENUS-9 Inaccurate Pending Rewards for VAI Vault Users

• Low ⓘ

Unresolved

ⓘ Alert

Marked as "Unresolved" by the client. The client provided the following explanation:

We would like to limit the scope for the upgrade, and therefore we will not fix the issue in this release

File(s) affected: `VAIVault.sol`

Description: The VAI vault's `pendingXVS()` function may not return the correct amount of XVS rewards the user is eligible for. This function uses `accXVSPerShare` to calculate the rewards, but `accXVSPerShare` is only updated in `updateVault()`. Therefore, the return of `pendingXVS()` is inaccurate between a `updatePendingRewards()` call and the first call that triggers `updateVault()`. Third-party protocols or front-ends calling `pendingXVS()` during this period will get a lower value than the actual.

Recommendation: Consider updating the `accXVSPerShare` variable in the `updatePendingRewards()` function. Ideally, the `updatePendingRewards()` and `updateVault()` can be combined into one function and executed at the beginning of each user operation. Whenever XVS rewards are sent to the VAI vault, the sender should execute the combined function in the same transaction.

VENUS-10 VRT Vault Interests Are Accrued During the Pausing Period

• Low ⓘ

Acknowledged

ⓘ Update

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

This is the intended behavior. Pauses are intended for emergency use only and should not affect users' rewards.

File(s) affected: `VRTVault.sol`

Description: Although the privileged roles can pause the deposit and withdrawals on the VRT vault, users still earn the interests during the pausing period since the interests are calculated block-by-block until the last accruing block. Users can claim their interest during this period when the vault is unpaused.

Recommendation: Confirm whether this is the intended behavior. If not, consider implementing a mechanism to exclude rewards during pausing.

VENUS-11 Inaccurate Calculation of Pending Rewards of XVS Vault Pools

• Low ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. Addressed in: `8d547ac656c33618125733e00d4ae237a16b82be` .

File(s) affected: XVSVault.sol

Description: According to the design of the XVS vault, pending withdrawals after the upgrade are not eligible for accruing rewards. However, the `pendingReward()` function includes every pending withdrawal when calculating the pending rewards. This issue would affect the front end and any third-party protocols integrating with the XVS vault using this function.

Recommendation: Consider modifying the `pendingReward()` function to exclude post-upgrade pending withdrawals when calculating pending rewards. As we pointed out in this report, the use of `balanceOf()` to get the current deposit of the specific pool may be an overestimation and should be avoided.

VENUS-12

Inaccurate Calculation of Stake Amount for XVS Vault Pools

• Low ⓘ Mitigated

i Update

Marked as "Mitigated" by the client. Addressed in: 4404d27c57d40bea97320a86437ba662e26b85ec . The client provided the following explanation:

We prohibited pools with the same staked tokens

File(s) affected: XVSVault.sol

Description: According to the code comments, the `getStakeAmount()` function

```
@notice Get the XVS stake balance of an account (excluding the pending withdrawals)
```

However, the returned stake amount only accounts for the pool whose pool and rewards tokens are both XVS. In other words, deposits in pools whose pool token is XVS but the rewards token is not XVS are not considered valid XVS stake balance.

Recommendation: Confirm whether this is the intended behavior. If not, consider modifying the logic to account for the deposits in every pool whose pool token is XVS.

VENUS-13 Vaults May Not Deliver Full Payouts

• Low ⓘ Mitigated

i Update

Marked as "Fixed" by the client. The contract now tracks any unpaid rewards a user is owed to pay them at a later date. However, given that there is no guarantee that the contract will reacquire the necessary liquidity to pay out rewards, we have marked the issue as "Mitigated". Addressed in: 78c97315c0850b9fe429e8475a66642ab04f2f4b . The client provided the following explanation:

Fixed by keeping track of undistributed debt in XVSVault: <https://github.com/VenusProtocol/venus-protocol/commit/78c97315c0850b9fe429e8475a66642ab04f2f4b> VAIVault, in turn, allocates rewards based on actual amount of XVS received from the Comptroller. The XVS is sent in an automated fashion, so the only scenario when the vault doesn't receive funding is if the rewards are exhausted in the Comptroller contract. Still, even with zero rewards, the vault will deliver the already allocated payouts.

File(s) affected: VAIVault.sol , XVSVault.sol , XVSSStore.sol

Description: The function `VAIVault.safeXVSTransfer()` transfers whatever XVS it can given the contract's balance, but `VAIVault._withdraw()` and `VAIVault.deposit()` both update the `user.rewardDebt` as if the full payout was delivered. `XVSSStore.safeRewardTransfer()` and its usages exhibit the same behavior.

Recommendation: Only increment `user.rewardDebt` by the amount transferred.

VENUS-14

Considerations on Claiming Rewards for Other Users for the VAI and XVS Vaults

• Low ⓘ Mitigated

i Update

Marked as "Mitigated" by the client. Addressed in: 78c97315c0850b9fe429e8475a66642ab04f2f4b . The client provided the following explanation:

The ability to claim for other users is intended as it simplifies the interaction with the protocol. Considering the threats mentioned, they are mitigated by the fix to #13.

File(s) affected: `XVSVault.sol`, `VAIVault.sol`

Description: The XVS Vault allows anyone to `claim()` a user's interest in any pool. Typically, this is not an issue when the pool has enough rewards to be claimed. However, the XVS vault is limited in its ability to distribute rewards by the number of rewards available in `XVSStore`. If there are not enough rewards at the time the staker claims, they will get fewer rewards than they should.

If the rewards are insufficient, stakers would rather wait until they become enough to claim. However, anyone can claim them and force them to lose some part of their rewards. A similar issue exists when claiming the XVS rewards in the VAI vault.

Recommendation: Confirm whether this is the intended behavior. If not, consider removing the function to disallow anyone from claiming rewards for another. If so, consider documenting the risk and impacts and inform the users with details. Also, consider implementing off-chain monitoring to ensure the rewards are sufficient for future blocks.

VENUS-15

A Compromised Admin Address Can Drain All of VRT

• Low ⓘ

Acknowledged

i Update

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

The admin of the VRT vault is Venus Governance. VRT is being deprecated, so having Governance access to VRTs makes sense – after the VRT swap is completed, the token will no longer have a use case, so the community will likely launch a vote to discontinue VRT vault and burn the remaining VRTs.

File(s) affected: `VRTVault.sol`

Description: Privileged users can call `withdrawBep20()` to transfer any amount of any token held by the contract to an address of their choice. While this is useful for transferring any unsupported tokens sent to the contract, this allows any privileged user to drain the contract of its entire VRT balance.

Recommendation: Do not allow `withdrawBep20()` to be called on the VRT token.

VENUS-16 Privileged Roles and Ownership

• Low ⓘ

Acknowledged

i Update

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

Access controls are configured via Venus Governance, so all the permission changes are voted upon. Venus team never submits proposals to grant any privileged roles to externally owned accounts, so private key leakage should not be a concern.

File(s) affected: `XVSVault.sol`, `XVSStore.sol`, `VRTVault.sol`, `VAIVault.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

Contracts `VAIVault.sol`, `VRTVault.sol` and `XVSVault.sol` had their access controls setup via the two custom contracts (which themselves were however out-of-scope for this audit) `AccessControlledV5.sol` and `AccessControlManager.sol`:

- Access is checked via function `AccessControlledV5._checkAccessAccessControlledV5Allowed()`, having a string of the function signature of the to-be-checked function as the parameter.
- Access is granted via `AccessControlManager.giveCallPermission()`, where addresses are granted access to specific function signature strings (and if non-zero, tied to specific contract addresses).
- Access is revoked via `AccessControlManager.revokeCallPermission()`, where addresses are revoked access to specific function signature strings (and if non-zero, tied to specific contract addresses).

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. Renounce the role (**and thereby preventing any future calls to the followingly listed functions!**) by calling `burnAdmin()`.
 2. Change the XVS store contract address (`xvsStore`) that controls reward token transfers by calling `setXvsStore()`.
 3. 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 the 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 denial 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() .
 1. **Funds could be held indefinitely by choosing high lockup times and/or always extending an existing period.**

The XVSSore.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 himself as an owner by calling setNewOwner() .
 3. Transfer an arbitrary amount by calling safeRewardTransfer() .

The VRTVault.sol contract contains the following privileged roles:

1. An administrator (admin), as initialized during the constructor execution to msg.sender :
 1. Initialize the contract (and defining the VRT token address and the interest per block) by calling initialize() .
 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() and withdraw()) by calling pause() .
 2. "resume()" : Resume the contract from a paused state by calling resume() .
 3. "withdrawBep20(address,address,uint256)" : **Withdraw an arbitrary amount of tokens from the contract by calling** withdrawBep20() .
 4. "setLastAccruingBlock(uint256)" : Modify at what block accrual should end by calling setLastAccruingBlock() .
 1. When calling it the first time it can be set arbitrarily in the future or the past.
 2. Once initialized but not yet reached, it can be set to an arbitrary block number in the future, or a value between the current block number and the current value of lastAccruingBlock .
 3. **Once reached/passed it, it can be changed to be at an arbitrary number in the past or the future and thereby re-enable** deposit() .

The VAIVault.sol contract contains the following privileged roles:

1. An administrator (admin , onlyAdmin modifier), as initialized during the constructor execution to msg.sender :
 1. Renounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling burnAdmin() .
 2. Assign a new admin by calling setNewAdmin() .
 3. Change the XVS and VAI token contract addresses by calling setVenusInfo() .
 1. **This function could also be abused by the admin to circumvent reentrancy guards.**

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(), withdraw() and updatePendingRewards()) by calling pause() .
2. "resume()" : Resume the contract from a paused state by calling resume() .

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

VENUS-17

XVS Vault's Domain Separator Does Not Include a Contract Version

• Low ⓘ

Unresolved

i Alert

Marked as "Unresolved" by the client. The client provided the following explanation:

We would like to avoid breaking changes to the functionality not directly related to this upgrade. Fixing this will likely be a part of a separate release.

File(s) affected: XVSVault.sol

Description: The domain separator calculated for the XVS vault does not include a `version` field. According to EIP-712, `string version` the current major version of the signing domain. Signatures from different versions are not compatible.

Without the `version` field, a signature signed for the current version of the contract can be reused in future versions as long as the current nonce does not change, which can be an issue since the contract may be upgraded again in the future.

Recommendation: Consider using OpenZeppelin's EIP-712 implementation, which includes the version field by default.

VENUS-18 Missing Input Validation

• Low ⓘ Mitigated

i Update

Marked as "Fixed" by the client. Given that the `XVSStore` was left unmodified, we have marked the issue as "Mitigated". Addressed in: `9048e507043aac4c488220401da1bfb61434a449`. The client provided the following explanation:

- We have added the following validations to XVSVault:
- address checks for `_rewardToken` and `_token` parameters in `add()`
 - address checks for `_xvs` and `_xvsStore` parameters in `setXvsStore()`
 - Regarding the rest of the mentioned validations:
 - We consider zero values for `_rewardPerBlock` and `_lockPeriod` parameters legitimate
 - `XVSStore` will not be changed in this release, so no updates to `XVSStore` contract were done

File(s) affected: XVSVault.sol, XVSStore.sol

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions do not have a proper validation of input parameters:

1. `XVSVault.add()` does not check:
 1. that parameters `_rewardToken` and `_token` are different from `address(0)`.
 2. that parameters `_rewardPerBlock` and `_lockPeriod` are non-zero.
2. `XVSVault.setXvsStore()` does not check:
 1. that parameters `_xvs` and `_xvsStore` are different from `address(0)`.
3. `XVSStore.setPendingAdmin()` does not check:
 1. that parameter `_admin` is different from `address(0)`.
 2. that parameter `_admin` is different from the previous admin address.
4. `XVSStore.setNewOwner()` does not check:
 1. that parameter `_owner` is different from the previous owner address.

Recommendation: We recommend adding the relevant checks.

VENUS-19 Use of Outdated Compiler Version

• Low ⓘ Acknowledged

i Update

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

Updating compiler version is out of the scope of this release

File(s) affected: contracts/*.sol

Description: The contracts are compiled under Solidity `v0.5.16`, which is considered outdated and may contain known compiler bugs that can affect the logic of the contracts.

The use of experimental features, e.g., `pragma experimental ABIEncoderV2;` should also be avoided since they were not tested thoroughly at the time of the outdated Solidity versions and several known bugs existed in the past.

Recommendation: Consider using a more recent or the latest Solidity version. Also, ensure no known compiler bugs under the chosen version can affect the contracts. A list of known compiler bugs can be found [here](#).

VENUS-20 Pool May Be Permanently Misconfigured

• Low ⓘ Fixed

✓ Update

Marked as "Fixed" by the client. Addressed in: `43a77f346a8cda5cc03e985ef66a1dd569f67e0f`

File(s) affected: `XVSVault.sol`

Description: The function `_updatePool()` divides by `totalAllocPoints[_rewardToken]` when calculating the reward. This variable is updated in the `add()` function:

```
totalAllocPoints[_rewardToken] = totalAllocPoints[_rewardToken].add(_allocPoint);
```

`_allocPoint` must be greater than zero the first time a pool is added. Otherwise, `_updatePool()` would revert due to division by zero, which would block users from calling `add()`, `deposit()`, `claim()`, and `requestWithdrawal()` for the corresponding `_rewardToken`. A blocked call to `massUpdatePools()` will prevent fixing the configuration by calling `set()`.

Recommendation: Require a non-zero allocation in `add()` and `set()`.

VENUS-21

VAI Losing 1\$ Peg May Lead to a Drained `VAIVault` and Destabilize the XVS Token and Vault

• Informational ⓘ Acknowledged

ⓘ Update

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

VAIVault distributions are continuously monitored by the risk team. The current value of reward per block corresponds to the market analysis. That being said, the reward amount per block is fixed, so users will not be receiving more XVS if there are more deposits.

File(s) affected: `VAIVault.sol`

Description: The [VAI token is an algorithmic stable coin](#) and pegged to 1 USD, similar to [MakerDAO's DAI](#). However, it is not guaranteed to always hold its peg to 1 USD under all circumstances.

Should the token lose its peg and be (much) less valuable than 1 USD it could lead to a high increase of users depositing their `VAI` tokens in the `VAIVault` and earning `XVS` tokens as interest at a (high) discount.

This could lead to an unexpectedly quick depletion of `XVS` funds in the `VAIVault` and in turn destabilize the `XVS` token value and all its associated pools in the `XVSVault`.

Recommendation: We recommend documenting this fact in public user-facing documentation, closely monitoring the `VAI` token, and, if necessary, pausing the `VAIVault`.

VENUS-22

Application Monitoring Can Be Improved by Emitting More Events

• Informational ⓘ Acknowledged

ⓘ Update

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

Improved monitoring is out of the scope for this release

File(s) affected: `VAIVault.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks.

There is currently no event indicating an update of `VAIVault`'s pending rewards.

Recommendation: Consider adding an event for `VAIVault.updatePendingRewards()`.

VENUS-23 Unlocked Pragma

• Informational ⓘ Acknowledged

Update

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

The pragma is locked in the contracts we're going to upgrade (XVSVault, VAIVault, VRTVault). Updating other contracts is out of the scope for this release.

File(s) affected: `contracts/*.sol`

Related Issue(s): [SWC-103](#)

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. 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.

VENUS-24 Vaults May Be Insufficiently Funded

• Undetermined ⓘ Mitigated

Update

Marked as "Mitigated" by the client. Addressed in: `78c97315c0850b9fe429e8475a66642ab04f2f4b`. The client provided the following explanation:

See #13

File(s) affected: `XVSVault.sol`, `VRTVault.sol`, `VAIVault.sol`

Description: All three vaults `VAIVault.sol`, `VRTVault.sol` and `XVSVault.sol` do not guarantee to hold sufficient funds at all times to cash out accrued rewards, as all three do not have automatic funding/minting mechanisms, but rather get funded manually by Venus at certain intervals.

In particular reward tokens used in the `XVSVault.sol` and `VAIVault.sol` contracts may pay out until the reward tokens supplies run dry, but will however take in the full deposited value (Due to `XVStore.safeRewardTransfer()` and `VAIVault.safeXVSTransfer()`, respectively).

Recommendation: We recommend documenting this fact and the anticipated funding intervals and amounts for all vaults in public user-facing documentation.

VENUS-25

`VRTVault` May Be Operated and Used Past the Last Accruing Block

• Undetermined ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `b0a896c88bef287c6a58e1fe51aa63e8cb4f2dab`.

File(s) affected: `VRTVault.sol`

Description: The `VRTVault` has a built-in deprecation through variable `lastAccruingBlock`, which dictates after which block number no more rewards will be accrued.

There are three scenarios when and how it can be changed:

1. When calling it the first time it can be set arbitrarily in the future or the past.
2. Once initialized but not yet reached, it can be set to an arbitrary block number in the future, or a value between the current block number and the current value of `lastAccruingBlock`.
3. **Once reached/passed it, it can be changed to be at an arbitrary number in the past or the future and thereby re-enable `deposit()`.**

Recommendation: We recommend clarifying if this is intentional and, if not, adding corresponding checks in `setLastAccruingBlock()` from changing `lastAccruingBlock` once the block has been reached.

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. The following typographical errors have been noted:
 1. `XVSVault.sol#L507`: `vairables` → `variables`.
 2. `VAIVault.sol#L233`: `owner` → `admin`.
2. Code comment `XVSVault.sol#L198` seems to be wrong and copy-and-pasted from `L188`. The corresponding function rather updates the withdrawal locking period.
3. Missing or incorrect NatSpec comments:
 1. Several externally/publicly callable functions are missing NatSpec comments entirely.
 2. `XVSVault.getEligibleWithdrawalAmount()`: Missing NatSpec comment for return value.
 3. `XVSVault.getRequestAmount()`: Missing NatSpec comment for return value.
 4. `XVSVault.getWithdrawalRequests()`: Missing NatSpec comment for return value.
 5. `XVSVault.pendingWithdrawalsBeforeUpgrade()`: Missing NatSpec comment for return value.
 6. `VRTVault.getAccruedInterest()`: Missing NatSpec comment for return value.
 7. `VRTVault.computeAccruedInterest()`: Missing NatSpec comment for return value.
 8. `VAIVault.pendingXVS()`: Missing NatSpec comment for return value.

Adherence to Best Practices

1. According to best practices it is advised to always use explicit type widths (i.e. `uint256` over `uint`). In this regard, consider the following cases:
 1. `XVSVault.sol#L45`.
 2. `XVSVault.sol#L51`.
 3. `XVSVault.sol#L54`.
 4. `XVSVault.sol#L59`.
 5. `XVSVault.sol#L61`.
 6. `XVSVault.sol#L62`.
 7. `XVSVault.sol#L63`.
 8. `XVSVault.sol#L67`.
 9. `XVSVault.sol#L281`.
 10. `XVSVault.sol#L282`.
 11. `XVSVault.sol#L284`.
 12. `XVSVault.sol#L312`.
 13. `XVSVault.sol#L315`.
 14. `XVSVault.sol#L387`.
 15. `XVSVault.sol#L388`.
 16. `XVSVault.sol#L413`.
 17. `XVSVault.sol#L424`.
 18. `XVSVault.sol#L449`.
 19. `XVSVault.sol#L455`.
 20. `XVSVault.sol#L613`.
 21. `XVSVault.sol#L614`.
 22. `XVSVault.sol#L681`.
 23. `XVSVault.sol#L686`.
2. For improved maintainability and gas-efficiency reasons, duplicate or otherwise unnecessary code should be avoided. In this regard, consider the following cases:

1. The subtraction of `sub(user.pendingWithdrawals)` in `XVSVault.sol#L224` and `L232` are redundant, as the corresponding value is already summed up and checked in `L221` (same applies for `L256` , `L261` and `L419` , `L428`).
2. The expression `user.amount.sub(user.pendingWithdrawals).mul(pool.accRewardPerShare).div(1e12)` in `XVSVault.sol#L224` and `L232` may be only computed once and then reused (same applies for `L256` , `L261` and `L419` , `L428`).
3. Struct variable `VRTVaultStorage.UserInfo.lastWithdrawnBlockNumber` remains unused.
3. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
 1. `XVSSStore.NewPendingAdmin()` ,
 2. `XVSVault.StoreUpdated()` ,
4. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead declare them once (as constant and commented) and use these constant variables instead. Following instances should therefore be changed accordingly:
 1. `XVSVault.sol#L224` , `L232` , `L256` , `L261` , `L358` , `L361` , `L419` , `L428` , `L502` , `L504` and `L537` : `1e12` .
 2. `VRTVault.sol#L198` : `1e18` .
 3. `VAIVault.sol#L101` , `L144` , `L156` and `L210` : `1e18` .
5. The number of reserved storage slots in contracts `VAIVaultStorage` , `VRTVaultStorage` , and `XVSVaultStorage` can be adjusted to make each contract occupy exactly 50 slots. (See [OpenZeppelin's documentation](#) for details)
6. Use `safeTransfer()` from `SafeBEP20` for the token transfer in the `VAIVault.safeXVSTransfer()` function.
7. In `XVSVault` , pushing and popping withdrawal requests may become gas intensive if a sufficiently large number of requests are made. A user may no longer be able to execute withdrawals if this occurs. Consider limiting the number of withdrawal requests a user can have at a time.

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

- `31e...891` ./XVSSStore.sol
- `292...7e8` ./XVSVault.sol
- `63b...2a6` ./XVSVaultStorage.sol
- `575...446` ./VRTVault.sol
- `cd2...477` ./VRTVaultStorage.sol
- `3dc...3c3` ./Vault/VAIVault.sol
- `fa0...a53` ./Vault/VAIVaultStorage.sol

Tests

- `3ab...144` ./vrtStopRewards.ts
- `523...81c` ./VRTVaultUpgrade.ts
- `7af...c77` ./XVSVaultUpgrade.ts
- `960...072` ./VAIVaultUpgrade.ts
- `95d...542` ./xvsVaultTest.js
- `12a...a20` ./VRTVault/vrtVaultTest.js

Automated Analysis

N/A

Test Suite Results

The test suite was run by calling `yarn test` . Only the relevant tests have been included in the output.

Update: The number of tests included increased significantly, particularly for `XVSVault` .

VAIVault

- ✓ claim reward (485ms)
- setVenusInfo
 - ✓ fails if called by a non-admin

- ✓ fails if XVS address is zero
- ✓ fails if VAI address is zero
- ✓ disallows configuring tokens twice (52ms)

VRTVault

unit tests

setLastAccruingBlock

- ✓ fails if ACM disallows the call
- ✓ fails if lastAccuringBlock has passed (52ms)
- ✓ fails if trying to set lastAccuringBlock to some past block
- ✓ fails if trying to set lastAccuringBlock to the current block
- ✓ correctly sets lastAccuringBlock to some future block (38ms)
- ✓ can move lastAccuringBlock to a later block (60ms)
- ✓ can move lastAccuringBlock to an earlier block (59ms)
- ✓ fails if trying to move lastAccuringBlock to a block in the past (49ms)

scenario

- ✓ deposit (93ms)
- ✓ should claim reward (60ms)
- ✓ should not claim reward after certain block (90ms)

XVSVault

setXvsStore

- ✓ fails if XVS is a zero address
- ✓ fails if XVSSStore is a zero address
- ✓ fails if the vault is already initialized

add

- ✓ reverts if ACM does not allow the call
- ✓ reverts if xvsStore is not set
- ✓ reverts if a pool with this (staked token, reward token) combination already exists (39ms)
- ✓ reverts if staked token exists in another pool
- ✓ reverts if reward token is a zero address
- ✓ reverts if staked token is a zero address
- ✓ emits PoolAdded event (42ms)
- ✓ adds a second pool to an existing rewardToken (56ms)
- ✓ sets pool info (54ms)
- ✓ configures reward token in XVSSStore (52ms)

setRewardAmountPerBlock

- ✓ reverts if ACM does not allow the call
- ✓ reverts if the token is not configured in XVSSStore (47ms)
- ✓ emits RewardAmountPerBlockUpdated event (50ms)
- ✓ updates reward amount per block (60ms)

pendingReward

- ✓ includes the old withdrawal requests in the rewards computation (198ms)
- ✓ excludes the new withdrawal requests from the rewards computation (237ms)

deposit

- ✓ reverts if the vault is paused (40ms)
- ✓ reverts if pool does not exist
- ✓ transfers pool token to the vault (91ms)
- ✓ updates user's balance (130ms)
- ✓ fails if there's a pre-upgrade withdrawal request (117ms)
- ✓ succeeds if the pre-upgrade withdrawal request has been executed (352ms)
- ✓ uses the safe _transferReward under the hood (208ms)

executeWithdrawal

- ✓ fails if the vault is paused
- ✓ only transfers the requested amount for post-upgrade requests (245ms)
- ✓ handles pre-upgrade withdrawal requests (248ms)
- ✓ handles pre-upgrade and post-upgrade withdrawal requests (366ms)

requestWithdrawal

- ✓ fails if the vault is paused (39ms)
- ✓ transfers rewards to the user (199ms)
- ✓ uses the safe _transferReward under the hood (199ms)
- ✓ fails if there's a pre-upgrade withdrawal request (97ms)

claim

- ✓ fails if there's a pre-upgrade withdrawal request (51ms)
- ✓ succeeds if the pre-upgrade withdrawal request has been executed (209ms)
- ✓ excludes pending withdrawals from the user's shares (317ms)
- ✓ correctly accounts for updates in reward per block (163ms)
- ✓ uses the safe _transferReward under the hood (110ms)

_transferReward

- ✓ sends the available funds to the user (90ms)
- ✓ emits VaultDebtUpdated event if vault debt is updated (63ms)
- ✓ does not emit VaultDebtUpdated event if vault debt is not updated (82ms)

- ✓ records the pending transfer (80ms)
 - ✓ records several pending transfers (148ms)
 - ✓ sends out the pending transfers in addition to reward if full amount <= funds available (251ms)
 - ✓ sends a part of the pending transfers and reward if full amount > funds available (229ms)
- pendingWithdrawalsBeforeUpgrade
- ✓ returns zero if there were no pending withdrawals
 - ✓ returns zero if there is only a new-style pending withdrawal (101ms)
 - ✓ returns the requested amount if there is an old-style pending withdrawal
 - ✓ returns the total requested amount if there are multiple old-style pending withdrawals (56ms)
 - ✓ returns zero if the pending withdrawal was executed (120ms)
- Scenarios
- ✓ works correctly with multiple claim, deposit, and withdrawal requests (773ms)

Code Coverage

Coverage was gathered by running `npx hardhat coverage` . Only the files in scope are included in the output. The line coverage for the vaults is between 57-71%. We recommend getting these values as close to 100% as possible.

Update: The line coverage has improved slightly to 64-78%, but still has room for improvement.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
VRTVault/	46.62	39.19	53.33	48.23	
VRTVault.sol	62	46.77	72.73	64.15	... 277,302,303
VRTVaultStorage.sol	100	100	100	100	
Vault/	49.49	47.5	51.85	50.49	
VAIVault.sol	77.78	59.38	73.68	78.79	... 217,218,236
VAIVaultStorage.sol	100	100	100	100	
XVSVault/	59.71	49.24	56.06	60.39	
XVSStore.sol	57.69	50	66.67	60.71	... 70,72,73,89
XVSVault.sol	67.36	52.78	63.27	68.04	... 814,820,821
XVSVaultStorage.sol	100	100	100	100	

Changelog

- 2023-05-19 - Initial report
- 2023-06-06 - Fix Review

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' 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.