CERTIK

# Preliminary Comments

# XBE Finance

Dec 31st, 2021

# Table of Contents

# Summary

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

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# Overview

## Project Summary

| | |
|---|---|
| Project Name | XBE Finance |
| Platform | ethereum |
| Language | Solidity |
| Codebase | https://github.com/XBEfinance/vault2-Certik/tree/branch_scope_1.10 |
| Commit | 1b8eedf9a460473217f97f3def8d9517dae1197e |

## Audit Summary

| | |
|---|---|
| Delivery Date | Dec 31, 2021 |
| Audit Methodology | Static Analysis, Manual Review |
| Key Components | |

## Vulnerability Summary

| Vulnerability Level | Total | Pending | Declined | Acknowledged | Partially Resolved | Resolved |
|---|---|---|---|---|---|---|
| Critical | 1 | 1 | 0 | 0 | 0 | 0 |
| Major | 21 | 21 | 0 | 0 | 0 | 0 |
| Medium | 9 | 9 | 0 | 0 | 0 | 0 |
| Minor | 10 | 10 | 0 | 0 | 0 | 0 |
| Informational | 40 | 40 | 0 | 0 | 0 | 0 |
| Discussion | 2 | 2 | 0 | 0 | 0 | 0 |

# Audit Scope

| ID | File | SHA256 Checksum |
|---|---|---|
| VIX | governance/utils/VotingInitializable.sol | f6613c74dc9ff04bfeacc695916ce25a3107675c8867f6c3a8f24fcdfefe8e42 |
| VNR | governance/utils/VotingNonReentrant.sol | bb1f0c21397f95098f03d2feaa86b96e6edc777c93d5c53ee4a89b3af92a516e |
| VOX | governance/utils/VotingOwnable.sol | 8a077b8f4c2127c96f9fd2f9308a40beec87f5f04d8b4285daf3a92e1f94a5f5 |
| VPX | governance/utils/VotingPausable.sol | 2a64fed2e8f92a80b3049e3bb6c869bf20865e550b7b5d017ee6e147251e14a0 |
| BMT | governance/BasicMetaTransaction.sol | 4b870d2ea2e3dec238ad89c7a31b27ff48426d94e196948ed72205bd03ccadec |
| EVM | governance/EVMScriptExecutorMock.sol | 62a01cfcc30d3d44e2f2a8fa9c82e0e3ac3999c33c76ebf6f9fe9751af2fb246 |
| VXB | governance/Voting.sol | dc0fc4b977a4313e60138b1308acf2bd30ec8b595027d0359a77829c063e9392 |
| VSR | governance/VotingStakingRewards.sol | 3368772f8042d5e2a8543967fe88718659e190f4cc4d3d3c28e7c5152a984737 |
| BXB | main/bank/Bank.sol | c9c7b854f0dfff0060ba2c190eaf6de719af8d1512916afe1938b7f021b8b946 |
| BPA | main/bank/BankProxyAdmin.sol | ab5d7252958e37a57e301206d5897dc88cf1803f1d4eb9f26944d3624403279f |
| BTP | main/bank/BankTransparentProxy.sol | ebf4179974efed6a7f443cca7e1d820bfac8d8cbf9e60cf9ac2722df57ffe777 |
| UTW | main/converters/UnwrappedToWrappedTokenConverter.sol | 5cdf2d4a922df40b8e45294a932e845830250f44a2ee0f4049c5ebb53bead6b3 |
| WTU | main/converters/WrappedToUnwrappedTokenConverter.sol | fdb31469f92cc835079394469c11d20fc28b7de38edab0bebfe8a1be5f7254b8 |
| RDR | main/staking_rewards/RewardsDistributionRecipient.sol | b238312a1e074bad8d6c5cf713e9dbeaf40dd344125aa5618c8cf8eac5599e25 |
| SRX | main/staking_rewards/StakingRewards.sol | adb6490ba5484b6d3a2ff98e489cd88498e541a5cd5c449d8827755da8e0f26e |
| BSX | main/strategies/base/BaseStrategy.sol | fa82ef6037396deae8cfe6269c83929fd11d4d443e9de3b1933c17358be858d0 |

| ID | File | SHA256 Checksum |
| --- | --- | --- |
| CSX | main/strategies/base/ClaimableStrategy.sol | 3af66ab53a6d1fb700b3ac4dda8d8d15681b2d0018e1e94a83a28ad8872ca267 |
| CVX | main/strategies/CVXStrategy.sol | 3e9b833ef89e5bb2cdf0190a5798e8b5a62015eafc012773d65093dfa8edc128 |
| CCS | main/strategies/CvxCrvStrategy.sol | 914135c3a1651acbc5fdb2fc15e6486e4721656627a3c77274ce4b1dbb5cef49 |
| HSX | main/strategies/HiveStrategy.sol | fcb3b65523eb1b539a7c30f3cb3aa6f4c4325952eb57479d858ef5ce1557acd7 |
| SSX | main/strategies/SushiStrategy.sol | 4849f35687cb79309193e330edea1e37d8349018bd8285dc881f0c10b0e9d103 |
| ERC | main/vaults/base/utils/ERC20Vault.sol | 501a398985f301dbc3d76d6e4272fdaf8b37c2cd534a289a341690d025ea010f |
| BVV | main/vaults/base/BaseVaultV2.sol | 2189aaeffa4979f51e2c447fdf7197b3cb5242767b0243adf09fbf0f37fac1f7 |
| VWA | main/vaults/base/VaultWithAutoStake.sol | c10f38b497805aac750a52f14cb4cb5b1504e9659304ccdfd380af2e1ec1526f |
| VWF | main/vaults/base/VaultWithFees.sol | 1e1fd39998398d4caae4ad7752e5ac7560b09e49f0a214c8d941e5c66a18f0b2 |
| VWR | main/vaults/base/VaultWithReferralProgram.sol | 13f7e4138e9dbe0d37fd8a37f4e7be7a28dcb9e972109bfc07b16bbd82993a2c |
| SVX | main/vaults/SushiVault.sol | 2f4a3a309fb570efbd8434ff5f1c5bb24dd5f3740b8f17b88d1f2dce14cb03a9 |
| VXE | main/vaults/Vault.sol | 909bd9a2c94e8db75a9ae7b5e1ce919f04e3f6cac56bb5e176df682d698daf74 |
| XCP | main/weth/weth9.sol | b1d5677686ffe359b63e29372eb541753e0066a3693c66c3e506ce72d47977f7 |
| BCX | main/BonusCampaign.sol | 61dcb9446898b2cf9a9e9177b8ea7cb8a9382b662d6b3cc641c8baf3178fed0f |
| CXB | main/Controller.sol | f0eeb65b6a75bff9e327bb68b1f42562b7197c5f971d9e4f1c9538f4fe094834 |
| FTT | main/FeeToTreasuryTransporter.sol | d7262d4f795f4435a3c42996c5ea6638cbdbeab471c3a96dcb06ae6f0545b09b |
| LSX | main/LockSubscription.sol | c361be4a88a7352789223bd6511c903d6e618ed9b49e225ca6fc9d48a5bd212e |

| ID | File | SHA256 Checksum |
| --- | --- | --- |
| RPX | main/ReferralProgram.sol | 25d30ab65c31476bcd497e879e2159727881401642dd5fe4af38fce0d22723b4 |
| RXB | main/Registry.sol | 484c132a912baf1a0da12343deaeb225b0cbc017c0db7b3e9a1f69b90e93abb9 |
| SXB | main/SimpleXBEInflation.sol | 3288e888d20a1e34242f3798a64cc12e9cb457fa6ecd6306149c72ca80812c59 |
| TWX | main/TokenWrapper.sol | ba8978560399cbe410dbcad82c9c19815f90eea7354324743e8e5493978f4604 |
| TXB | main/Treasury.sol | 07f2302244d7674c0c8f5963626ecccfc3148b4c3b44425708cd970fc7593f45 |
| VXX | main/VeXBE.sol | a6c256ba82c256210f89a7a57c4780902d55f13b7f7aff125c934b7b23af360e |

# Review Notes

## Overview

XBE Finance is a protocol within decentralized finance that contains a variety of finance implementations. Some of their major products are yield aggregators, such as the XBE Hive. Users are able to stake tokens to earn yields in a variety of tokens such as XBE, CRV, and CVX. In addition, users can stake and lock their XBE tokens to acquire voting rights.

XBE Finance also includes ways to incorporate users within traditional finance to participate with their decentralized finance programs.

## Dependencies

There are a few depending injection contracts or addresses in the current project:

- `token` and addresses in `_executionScript` for the contract `Voting`;
- `rewardsToken`, `stakingToken`, `token`, and `boostLogicProvider` for the contract `VotingStakingRewards`;
- `eurxb` for the contract `Bank`;
- `token`, `_strategy`, and `wrapper` for the contract `UnwrappedToWrappedTokenConverter`;
- `token`, `_strategy`, and `wrapper` for the contract `WrappedToUnwrappedTokenConverter`;
- `rewardsToken` and `stakingToken` for the contract `StakingRewards`;
- `_want`, `controller`, and `_token` for the contract `BaseStrategy`;
- `controller`, `_rewardToken`, and `_vault` for the contract `ClaimableStrategy`;
- `cvxRewards` and `wantToken` for the contract `CVXStrategy`;
- `crvRewards`, `cvxRewards`, `convexBooster`, `crvDepositor`, `crvToken`, `cvxCrvToken`, `curveCvxCrvStableSwapPool`, `wantToken`, `controller`, and `vault` for the contract `CvxCrvStrategy`;
- `crvRewards`, `cvxRewards`, `convexBooster`, and `wantToken` for the contract `HiveStrategy`;
- `_controller`, `stakingToken`, all addresses in `_validTokens`, and `_controller.strategies(address(stakingToken))` for the contract `BaseVaultV2`;
- `votingStakingRewards`, `tokenToAutostake`, and `_token` for the contract `VaultWithAutoStake`;
- `_rewardToken`, `_claimFee.to`, and `_stakingToken` for the contract `VaultWithFees`;
- `referralProgram` for the contract `VaultWithReferralProgram`;
- `rewardsToken` and `veToken` for the contract `BonusCampaign`;
- `_token`, `_strategy`, `strategies[_token]`, `strategies[_wantToken]`, `_vault`, `_want`, and `converter` for the contract `Controller`;

- `uniswapRouter`, `rewardsToken`, and all addresses in `_tokensToConvert` for the contract `FeeToTreasuryTransporter`;
- all addresses in `subscribers` for the contract `LockSubscription`;
- `registry` and all addresses in `tokens` for the contract `ReferralProgram`;
- `_vault`, `controller`, `strategy`, and all addresses in `_vaults` for the contract `Registry`;
- `token` for the contract `SimpleXBEInflation`;
- `wrappedToken` for the contract `TokenWrapper`;
- `uniswapRouter`, `rewardsDistributionRecipientContract`, `rewardsToken`, all addresses in `_tokensToConvert`, and `_tokenAddress` for the contract `Treasury`;
- `votingStakingRewards` and `registrationMediator` for the contract `VeXBE`.

We assume these contracts or addresses are valid and non-vulnerable actors and implement proper logic to collaborate with the current project.

## Privileged Functions

### Voting

In the contract `Voting`, the role `MODIFY_SUPPORT_ROLE` has the authority over the following function:

- `Voting.changeSupportRequiredPct()`, which changes the amount of `yea` votes versus `nay` votes a proposal needs to pass.

In addition, the role `MODIFY_QUORUM_ROLE` has authority over the following function:

- `Voting.changeMinAcceptQuorumPct()`, which changes the amount of `yea` votes versus total possible votes a proposal needs to pass.

Also, the role `CREATE_VOTES_ROLE` has authority over the following functions:

- `Voting.newVote(bytes,string)`, which creates a new proposal, votes for the caller in support of the proposal, and executes the proposal if it meets the voting requirements;
- `Voting.newVote(bytes,string,bool,bool)`, which creates a new proposal, may vote for the caller, and may try to execute the proposal;
- `Voting.forward()`, which creates a new proposal, votes for the caller in support of the proposal, and executes the proposal if it meets the voting requirements.

### VotingStakingRewards

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

- `VotingStakingRewards.setRewardsDistribution()`, which changes the address of `rewardsDistribution`;
- `VotingStakingRewards.setInverseMaxBoostCoefficient()`, which changes the minimum boost level;
- `VotingStakingRewards.setPenaltyPct()`, which changes the amount of penalty applied to premature bonded withdrawals;
- `VotingStakingRewards.setBondedLockDuration()`, which changes how long bonded stakes are locked up for;
- `VotingStakingRewards.setBoostLogicProvider()`, which changes the address of `boostLogicProvider`;
- `VotingStakingRewards.setAddressWhoCanAutoStake()`, which decides whether an address can stake for others or not;
- `VotingOwnable.renounceOwnership()`, which disables all functions that can only be called by `_owner`;
- `VotingOwnable.transferOwnership()`, which changes the address of `_owner`.

In addition, the role `rewardsDistribution` has authority over the following function:

- `VotingStakingRewards.notifyRewardAmount()`, which updates the rewards to be provided.

The role `pauser` has authority over the following functions:

- `VotingPausable.setPaused()`, which can disable or re-enable the functions `stakeFor()` and `stake()`;
- `VotingPausable.transferOwnership()`, which changes the address of `pauser`.

### VeXBE

In the contract `VeXBE`, the role `admin` has the authority over the following functions:

- `VeXBE.setMinLockDuration()`, which sets the minimum lock duration;
- `VeXBE.setVoting()`, which sets the `votingStakingRewards` address;
- `VeXBE.commitTransferOwnership()`, which sets the future `admin` candidate address;
- `VeXBE.applyTransferOwnership()`, which sets the future `admin` candidate to be `admin`.

### TokenWrapper

In the contract `TokenWrapper`, the role `MINTER_ROLE` has the authority over the following functions:

- `TokenWrapper.mint(uint256)`, which transfers `wrappedtoken` tokens from `MINTER_ROLE` to the contract and mints the same amount of tokens to `MINTER_ROLE`;

- `TokenWrapper.burn()`, which burns an amount of tokens for `MINTER_ROLE` and transfers the same amount of `wrappedtoken` tokens to `MINTER_ROLE`;
- `ERC20PresetMinterPauser.mint(address,uint256)`, which mints an amount of tokens to an address.
- `AccessControl.renounceRole()`, which removes the caller's `MINTER_ROLE` authority.

In addition, the role `PAUSER_ROLE` has the authority over the following functions:

- `ERC20PresetMinterPauser.pause()`, which disables token transfers, mints, and burns;
- `ERC20PresetMinterPauser.unpause()`, which re-enables token transfers, mints, and burns.
- `AccessControl.renounceRole()`, which removes the caller's `PAUSER_ROLE` authority.

In addition, the role `DEFAULT_ADMIN_ROLE` has the authority over the following functions:

- `AccessControl.grantRole()`, which grants a role to an address;
- `AccessControl.revokeRole()`, which revokes a role from an address;
- `AccessControl.renounceRole()`, which removes the caller's `DEFAULT_ADMIN_ROLE` authority.

### StakingRewards

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

- `StakingRewards.pause()`, which prevents users from staking;
- `StakingRewards.unpause()`, which allows users to stake again;
- `RewardsDistributionRecipient.setRewardsDistribution()`, which changes the address for `rewardsDistribution`;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `rewardsDistribution` has the authority over the following function:

- `StakingRewards.notifyRewardAmount()`, which updates the reward available.

### CVXStrategy

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

- `CVXStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and the `cvxRewards`;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;

- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `CVXStrategy.deposit()`, which stakes the `_want` tokens in the contract to `cvxRewards`;
- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

### CvxCrvStrategy

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

- `CvxCrvStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and various information for the `Cvx` pool;
- `CvxCrvStrategy.setPoolIndex()`, which decides which `Cvx` pool the strategy will be using;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `CvxCrvStrategy.deposit()`, which deposits the `_want` tokens in the contract to the associated `Cvx` pool;
- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

### HiveStrategy

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

- `HiveStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and various information for the `Cvx` pool;
- `HiveStrategy.setPoolIndex()`, which decides which `Cvx` pool the strategy will be using;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `HiveStrategy.deposit()`, which deposits the `_want` tokens in the contract to the associated `Cvx` pool;
- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

### SushiStrategy

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

- `SushiStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and various information for the liquidity pool;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following function:s

- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

### Controller

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

- `Controller.configure()`, which sets the addresses for the `_treasury`, the `strategist`, and transfers ownership to `_governance`;
- `Controller.inCaseTokensGetStuck()`, which transfers tokens in the contract to `_owner`;
- `Controller.inCaseStrategyTokenGetStuck()`, which transfers non-want tokens in a strategy contract to this contract;
- `Controller.setWithdrawAbility()`, which decides which addresses can withdraw a strategy's tokens to its associated vault;
- `Controller.setTreasury()`, which decides the address for `_treasury`;
- `Controller.setStrategist()`, which decides the address for `strategist`;
- `Controller.setVault()`, which decides the vault address associated to a token;
- `Controller.setConverter()`, which decides the address for a token convertor;
- `Controller.setStrategy()`, which sets an approved strategy for a token;
- `Controller.setApprovedStrategy()`, which decides if a strategy is approved or not for a token;

- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `strategist` has the authority over the following functions:

- `Controller.inCaseStrategyTokenGetStuck()`, which transfers non-want tokens in a strategy contract to this contract;
- `Controller.setVault()`, which decides the vault address associated to a token;
- `Controller.setConverter()`, which decides the address for a token convertor;
- `Controller.setStrategy()`, which sets an approved strategy for a token.

Addresses with `canWithdraw[address] = true` also have authority over the following function:

- `Controller.withdraw()`, which transfers a strategy's `_want` tokens to its associated vault.

### BonusCampaign

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

- `BonusCampaign.configure()`, which initializes the contract and sets the values for important variables;
- `BonusCampaign.setRegistrator()`, which decides the address for `registrator`;
- `BonusCampaign.startMint()`, which mints the rewards;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `registrator` has the authority over the following function:

- `BonusCampaign.processLockEvent()`, which registers a valid account to receive the maximum boost.

### FeeToTreasuryTransporter

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

- `FeeToTreasuryTransporter.addTokenToConvert()`, which includes a token to be converted into reward tokens;
- `FeeToTreasuryTransporter.removeTokenToConvert()`, which removes a token to be converted into rewards tokens;
- `FeeToTreasuryTransporter.setRewardsToken()`, which decides the address for the reward token;
- `FeeToTreasuryTransporter.setTreasury()`, which sets the address for the treasury;
- `FeeToTreasuryTransporter.setUniswapRouter()`, which sets the address for the Uniswap contract;

- `FeeToTreasuryTransporter.sendRewardToTreasure()`, which transfers reward tokens in the contract to the treasury;
- `FeeToTreasuryTransporter.convertToRewardsToken()`, which swaps tokens in the contract for reward tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

### LockSubscription

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

- `LockSubscription.setEventSource()`, which sets the address for `eventSource`;
- `LockSubscription.addSubscriber()`, which adds an address to the set of subscribers;
- `LockSubscription.removeSubscriber()`, which removes an address from the set of subscribers;
- `LockSubscription.pause()`, which prevents `processLockEvent()` from being called;
- `LockSubscription.unpause()`, which re-enables the `processLockEvent()` function;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `eventSource` has the authority over the following function:

- `LockSubscription.processLockEvent()`, which processes a lock event for an account at each subscriber.

### ReferralProgram

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

- `ReferralProgram.changeDistribution()`, which changes how fees are distributed among a user's chain of referrals;
- `ReferralProgram.addNewToken()`, which adds a new token to be considered for collecting fees;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, any `controller` for a registered vault has the authority over the following functions:

- `ReferralProgram.registerUser()`, which registers a user and sets who their referral is;
- `ReferralProgram.feeReceiving()`, which distributes a fee among the chain of referrals for a user.

### Registry

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

- `Registry.addVault()`, which adds a vault and its controller to the registry;

- `Registry.addWrappedVault()`, which adds a wrapped vault, its controller, and its unwrapped vault to the registry;
- `Registry.addDelegatedVault()`, which adds a delegated vault and its controller to the registry;
- `Registry.removeVault()`, which removes a vault from the registry;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

### SimpleXBEInflation

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

- `SimpleXBEInflation.configure()`, which decides the token to mint, the maximum amount to mint, and the amount to mint in each period;
- `SimpleXBEInflation.setXBEReceiver()`, which allows an address to receive minted tokens and decides how much to receive;
- `SimpleXBEInflation.removeXBEReceiver()`, which decides which address will no longer receive minted tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

### Treasury

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

- `Treasury.configure()`, which decides the information for rewards, for swapping tokens, and also transfers ownership to `_governance`;
- `Treasury.setRewardsToken()`, which decides the address of the rewards token;
- `Treasury.setSlippageTolerance()`, which decides the amount of slippage in token swaps;
- `Treasury.setRewardsDistributionRecipientContract()`, which decides the address of the contract that distributes rewards;
- `Treasury.setAuthorized()`, which decides which addresses are authorized to convert tokens;
- `Treasury.addTokenToConvert()`, which adds a token to be converted to rewards tokens;
- `Treasury.removeTokenToConvert()`, which removes a token to be converted;

- `Treasury.toGovernance()`, which transfers any token in the contract to `_owner`;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, addresses that are authorized have access to the following function:

- `Treasury.convertToRewardsToken()`, which swaps a token to the rewards token via Uniswap.

### SushiVault

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

- `SushiVault.configure()`, which initializes important information like the staking token, rewards tokens, rewards duration, the controller, and transfers ownership to `_governance`;
- `BaseVaultV2.setTrustworthyEarnCaller()`, which decides the address for `trustworthyEarnCaller`;
- `BaseVaultV2.setController()`, which decides the address for `_controller`;
- `BaseVaultV2.setRewardsDistribution()`, which decides the address for `rewardsDistribution`;
- `BaseVaultV2.pause()`, which prevents deposits;
- `BaseVaultV2.unpause()`, which re-enables deposits;
- `BaseVaultV2.setRewardsDuration()`, which decides the amount of time for a reward to be paid out;
- `BaseVaultV2.addRewardToken()`, which adds a token that can be given as rewards;
- `BaseVaultV2.removeRewardToken()`, which removes a token that can be given as rewards;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `rewardsDistribution` has the authority over the following function:

- `BaseVaultV2.notifyRewardAmount()`, which adds a reward to be distributed.

The role `trustworthyEarnCaller` also has the authority over the following function:

- `BaseVaultV2.earn()`, which transfers staking tokens to the controller and claims any reward tokens.

### Vault

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

- `Vault.configure()`, which initializes important information like the staking token, rewards tokens, rewards duration, the controller, and transfers ownership to `_governance`;
- `VaultWithFees.setFeesEnabled()`, which decides if fees are applied to rewards or not;

- `VaultWithFees.addClaimFeeReceiver()`, which adds an address that receives fees and decides the percentage it receives;
- `VaultWithFees.removeClaimFeeReceiver()`, which removes an address that receives fees;
- `VaultWithFees.setClaimFeePercentage()`, which changes the percentage of fees sent to a receiver;
- `VaultWithFees.setDepositFee()`, which changes the percentage of fees applied to deposits;
- `VaultWithFees.setDepositWallet()`, which changes where deposit fees are sent to;
- `BaseVaultV2.setTrustworthyEarnCaller()`, which decides the address for `trustworthyEarnCaller`;
- `BaseVaultV2.setController()`, which decides the address for `_controller`;
- `BaseVaultV2.setRewardsDistribution()`, which decides the address for `rewardsDistribution`;
- `BaseVaultV2.pause()`, which prevents deposits;
- `BaseVaultV2.unpause()`, which re-enables deposits;
- `BaseVaultV2.setRewardsDuration()`, which decides the amount of time for a reward to be paid out;
- `BaseVaultV2.addRewardToken()`, which adds a token that can be given as rewards;
- `BaseVaultV2.removeRewardToken()`, which removes a token that can be given as rewards;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `rewardsDistribution` has the authority over the following function:

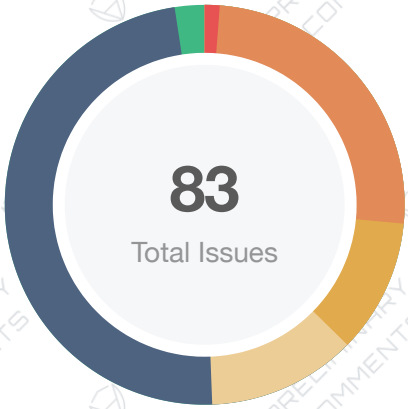
- `BaseVaultV2.notifyRewardAmount()`, which adds a reward to be distributed.

The role `trustworthyEarnCaller` also has the authority over the following function:

- `BaseVaultV2.earn()`, which transfers staking tokens to the controller and claims any reward tokens.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

# Findings




| | |
|---|---|
| Critical | 1 (1.20%) |
| Major | 21 (25.30%) |
| Medium | 9 (10.84%) |
| Minor | 10 (12.05%) |
| Informational | 40 (48.19%) |
| Discussion | 2 (2.41%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| GLOBAL-01 | Configure or Initialize Functions Lack Restrictions | Logical Issue | Minor | Pending |
| GLOBAL-02 | Unlocked Compiler Version | Language Specific | Informational | Pending |
| **BCX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| BCX-02 | Potential Reentrancy Attack | Logical Issue | Medium | Pending |
| BCX-03 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| BMT-01 | Potential Short Address Attack | Volatile Code | Minor | Pending |
| BMT-02 | Encoding of `msg.data` May Cause Incorrect Return Values | Language Specific | Minor | Pending |
| BSX-01 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| BVV-01 | Potential Error When `stakingToken` Is the Same As `_rewardsToken` | Logical Issue | Informational | Pending |
| BVV-02 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| BVV-03 | Possible for Rewards to Be Stuck if There Are No Stakers | Logical Issue | Informational | Pending |
| BVV-04 | Function Visibility Optimization | Gas Optimization | Informational | Pending |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| BXB-01 | Bank Deposits and Interest Are Not Updated During Transfers | Logical Issue | Critical | Pending |
| BXB-02 | Potential Reentrancy Attack | Logical Issue | Medium | Pending |
| BXB-03 | Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Calls | Volatile Code | Minor | Pending |
| **CCS-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| CCS-02 | Inconsistent Logic for `_want` Token | Logical Issue | Major | Pending |
| CCS-03 | Movement of Funds After Changing `poolIndex` | Logical Issue | Medium | Pending |
| CCS-04 | Unhandled Return Values | Volatile Code | Minor | Pending |
| CCS-05 | Potential Sandwich Attacks | Logical Issue | Informational | Pending |
| CCS-06 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **CVX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| **CXB-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| CXB-02 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **FTT-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| FTT-02 | Potential Sandwich Attacks | Logical Issue | Informational | Pending |
| FTT-03 | Unchecked Value of ERC-20 `transfer()` Call | Volatile Code | Minor | Pending |
| FTT-04 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **HSX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| HSX-02 | Movement of Funds After Changing `poolIndex` | Logical Issue | Medium | Pending |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| HSX-03 | Unhandled Return Values | Volatile Code | Minor | Pending |
| HSX-04 | Unused Variable | Gas Optimization | Informational | Pending |
| HSX-05 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **LSX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| LSX-02 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| LSX-03 | Function Visibility Optimization | Gas Optimization | Informational | Pending |
| RDR-01 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **RPX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| RPX-02 | Unable to Remove Tokens | Logical Issue | Informational | Pending |
| **RXB-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| **SRX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| SRX-02 | Possible for Rewards to Be Stuck if There Are No Stakers | Logical Issue | Informational | Pending |
| SRX-03 | Potential Error When `stakingToken` Is the Same As `rewardsToken` | Logical Issue | Informational | Pending |
| **SSX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| SSX-02 | Unimplemented Functions in `SushiStrategy` | Logical Issue | Discussion | Pending |
| **SVX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| **SXB-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| SXB-02 | Incorrect Weight When Adding a Removed `_xbeReceiver` | Logical Issue | Medium | Pending |
| SXB-03 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **TWX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| TWX-02 | Function Visibility Optimization | Gas Optimization | Informational | Pending |
| **TXB-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| TXB-02 | Unhandled Return Values | Logical Issue | Minor | Pending |
| TXB-03 | Potential Sandwich Attacks | Logical Issue | Informational | Pending |
| TXB-04 | Slippage Tolerance is Not Used | Inconsistency | Informational | Pending |
| TXB-05 | Rewards Are Not Updated Prior to Changing Rewards Token | Logical Issue | Informational | Pending |
| TXB-06 | Function Visibility Optimization | Gas Optimization | Informational | Pending |
| VPX-01 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **VSR-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| VSR-02 | Calculation of Rewards Uses Current Boost Level | Logical Issue | Major | Pending |
| VSR-03 | Inaccurate Boost Level Calculation | Inconsistency | Medium | Pending |
| VSR-04 | Staking Tokens Are Transferred to Treasury Instead of Reward Tokens | Logical Issue | Informational | Pending |
| VSR-05 | Bypassing Bonded Unlock Times | Logical Issue | Informational | Pending |
| VSR-06 | Potential Error When `stakingToken` Is the Same As `rewardsToken` | Logical Issue | Informational | Pending |
| VSR-07 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| VSR-08 | Possible for Rewards to Be Stuck if There Are No Stakers | Logical Issue | Informational | Pending |
| VSR-09 | No Restriction on `bondedLockDuration` | Volatile Code | Informational | Pending |
| VWF-01 | Fee Distribution Inconsistent With Documents | Inconsistency | Medium | Pending |
| VWF-02 | Possible Integer Overflow | Volatile Code | Minor | Pending |
| VWF-03 | `sumClaimFee` Lacks an Upper Bound | Volatile Code | Minor | Pending |
| VWF-04 | Uninitialized Storage Variable | Language Specific | Informational | Pending |
| VWF-05 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| **VXB-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| VXB-02 | Potential Reentrancy Attack | Logical Issue | Medium | Pending |
| VXB-03 | Function Visibility Optimization | Gas Optimization | Informational | Pending |
| **VXE-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| **VXX-01** | Centralization Risk | **Centralization / Privilege** | **Major** | Pending |
| VXX-02 | Anyone Can Allow Themselves to Create Locks for Others | Logical Issue | Medium | Pending |
| VXX-03 | Check Staking Amount Near the Beginning of Function Call | Gas Optimization | Informational | Pending |
| VXX-04 | Zero Voting Power For Small Locked Amount | Volatile Code | Informational | Pending |
| VXX-05 | Lack of Event Emissions for Significant Transactions | Coding Style | Informational | Pending |
| VXX-06 | Function Visibility Optimization | Gas Optimization | Informational | Pending |
| VXX-07 | `votingStakingRewards` Never Calls `createLockFor` | Logical Issue | Discussion | Pending |

# GLOBAL-01 | Configure or Initialize Functions Lack Restrictions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | Global | Pending |

## Description

The following contracts have configure or initialize functions that can be called by anyone:

- `Voting`
- `VotingStakingRewards`
- `Bank`
- `UnwrappedToWrappedTokenConverter`
- `WrappedToUnwrappedTokenConverter`
- `FeeToTreasuryTransporter`
- `ReferralProgram`
- `VeXBE`

This allows attackers to potentially front-run initialization transactions and manipulate sensitive variables.

## Recommendation

We recommend placing restrictions on who is able to properly initialize these contracts.

# GLOBAL-02 | Unlocked Compiler Version

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Language Specific | Informational | Global | Pending |

## Description

All contracts have an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

## Recommendation

We advise that the compiler version is instead locked at the lowest possible version the contract can be compiled at. For example, for version `v0.6.2` the contract should instead contain the following line of code:

```
pragma solidity 0.6.2;
```

## BCX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/BonusCampaign.sol (79bf1fe): 22, 46, 51, 68 | Pending |

### Description

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

- `BonusCampaign.configure()`, which initializes the contract and sets the values for important variables;
- `BonusCampaign.setRegistrator()`, which decides the address for `registrator`;
- `BonusCampaign.startMint()`, which mints the rewards;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `registrator` has the authority over the following function:

- `BonusCampaign.processLockEvent()`, which registers a valid account to receive the maximum boost.

Any compromise to the `_owner` or `registrator` account may allow the hacker to take advantage of this and disrupt rewards for early adopters.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` and `registrator` accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# BCX-02 | Potential Reentrancy Attack

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/main/BonusCampaign.sol (79bf1fe): 60 | Pending |

## Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run before the external call resolved its effects.

The function `startMint()` makes an external call to `rewardsToken` before state updates. This allows `rewardsToken` to reenter and call `startMint()` again, potentially minting more tokens than expected.

## Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin's ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

# BCX-03 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/BonusCampaign.sol (79bf1fe): 46 | Pending |

## Description

The following function updates the status of sensitive state variables and should be able to emit events as notifications:

- `setRegistrator()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned function.

# BMT-01 | Potential Short Address Attack

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/governance/BasicMetaTransaction.sol (79bf1fe) : 98 | Pending |

## Description

An address in Ethereum is meant to be 20 bytes long. A short address attack occurs when less than 20 bytes are given as an input for an address, causing the EVM to pad the end of function arguments with zeroes.

Depending on how this function is utilized, a short address attack may cause incorrect function arguments.

## Recommendation

We recommend adding a validity check on the size of `msg.data.length`.

# BMT-02 | Encoding of `msg.data` May Cause Incorrect Return Values

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Language Specific | Minor | projects/XBE-finance/contracts/governance/BasicMetaTransaction.sol (79bf1fe): 95 | Pending |

## Description

In solidity version 0.4.24, depending on the ABI encoding function used, `msgSender()` may return an unexpected value.

The encoding `abi.encodePacked()` pads zeroes on the right of inputs to reach 32 bytes. For instance, if `msgSender` is called by itself with `msg.data` containing `abi.encodePacked(bytes4(keccak256("msgSender()")), addr)` and `addr` is a valid address, then it will return the last 8 bytes of addr followed by 12 bytes of zeroes. This issue may occur if msgSender() is called from executeMetaTransaction()`.

The encodings `abi.encodeWithSelector()` and `abi.encodeWithSignature()` pads zeroes on the left of inputs to reach 32 bytes, but they also contain a bug where an extra 28 bytes of zeroes is padded on the right, causing `msgSender()` to always return the zero address if `msg.data` was created through these encodings.

The encoding `abi.encode()` pads zeroes on the left of inputs to reach 32 bytes and will return the correct address when passed through `msgSender()`.

## Recommendation

We recommend being careful about which ABI encoding function is used when working with `msgSender()`.

# BSX-01 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/strategies/base/BaseStrategy.sol (79bf1fe): 63, 70 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setController()`
- `setWant()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

## BVV-01 | Potential Error When `stakingToken` Is the Same As `_rewardsToken`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/vaults/base/BaseVaultV2.sol (79bf1fe): 375 | Pending |

### Description

In the function `notifyRewardAmount()`, there is a check to ensure that the contract's rewards token balance does not exceed the provided reward.

```
375         uint256 balance = rewardsToken.balanceOf(address(this));
376         require(
377             rewardRate <= balance.div(rewardsDuration),
378             "Provided reward too high"
379         );
```

However, it is possible for the staking token to be the same as the rewards token. In this case, `notifyRewardAmount()` may provide a reward that includes users' deposits, meaning that users will be unable to withdraw their deposits if it has already been given out as rewards.

### Recommendation

If the staking token and the reward token are the same, we recommend having the `rewardsDistribution` be a contract that always transfers the correct reward amount when calling `notifyRewardAmount()`.

# BVV-02 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/vaults/base/BaseVaultV2.sol (79bf1fe) : 133, 139, 144, 165, 169 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setTrustworthyEarnCaller()`
- `setController()`
- `setRewardsDistribution()`
- `addRewardToken()`
- `removeRewardToken()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

# BVV-03 | Possible for Rewards to Be Stuck if There Are No Stakers

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/vaults/base/BaseVaultV2.sol (79bf1fe): 189 | Pending |

## Description

The amount of rewards that can be distributed is based on `rewardPerTokenStored`, which can only increase if `totalSupply > 0`.

Hence if there are no stakers, then `rewardPerTokenStored` does not increase, but `lastUpdateTime` can increase, such as when `notifyRewardAmount()` or `updateReward()` is called.

If `T` represents the duration of when `totalSupply = 0`, then `rewardRate * T` amount of rewards will be forever lost.

## Recommendation

We recommend implementing a way for these lost rewards to be added to `rewardPerTokenStored`.

# BVV-04 | Function Visibility Optimization

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/vaults/base/BaseVaultV2.sol (79bf1fe): 240, 250, 260, 295, 299, 343, 451 | Pending |

## Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope:

- `deposit()`
- `depositFor()`
- `depositAll()`
- `withdraw()`
- `withdrawAll()`
- `getReward()`
- `balance()`

The functions that are never called internally within the contract should have external visibility.

## Recommendation

We advise that the functions' visibility specifiers are set to `external`, optimizing the gas cost of the functions.

# BXB-01 | Bank Deposits and Interest Are Not Updated During Transfers

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Critical | projects/XBE-finance/contracts/main/bank/Bank.sol (79bf1fe): 11 | Pending |

## Description

The contract `Bank` inherits the `ERC20` contract, which contains token transfer functions, allowing a user to transfer as many tokens as `_balances[user]` contains. However, transferring tokens does not change `deposits[user]` or collect interest, which may cause unintended consequences.

For example, if user A transferred `bEURxb` tokens to user B, user A would be unable to call `Bank.withdraw()` on the entirety of `deposits[A]` due to the `_burn` function in `Bank.withdraw()`. In addition, user B would also be unable to call `Bank.withdraw()` on their received tokens since `Bank.withdraw()` checks `deposits[B]` and not `_balances[B]`.

Another consequence is that interest is calculated on a user's `deposits[user]`. Hence a malicious user may deposit `EURxb` tokens, convince a victim to buy their `bEURxb` tokens, but still collect interest on their original `EURxb` tokens deposit while the victim will be unable to withdraw or collect interest on their newly acquired `bEURxb` tokens.

## Recommendation

We recommend updating a sender's and receiver's `_balances` amount when transferring tokens or using `_balances` instead of `deposits`. We also suggest updating interest amounts for the sender and receiver before a transfer occurs.

# BXB-02 | Potential Reentrancy Attack

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/main/bank/Bank.sol (79bf1fe): 77, 96, 108 | Pending |

## Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would otherwise not have run before the external call resolved its effects.

In the function `withdraw()`, a user's deposits and index are updated after calls to `eurxb.accrueInterest()` and `eurxb.transfer()`. Then if an attacker calls `withdraw()`, the interest on their deposit is calculated and if they re-enter during the transfer, their interest will be calculated again using the old deposits and index, allowing them to claim double the intended interest.

The above reasoning also applies to the functions `withdrawInterestEUR()` and `withdrawInterestBank()`, where the caller's index is updated after accruing interest or a transfer, allowing an attacker to re-enter and receive the interest multiple times.

## Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin's ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

## BXB-03 | Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Calls

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/main/bank/Bank.sol (79bf1fe): 54, 87, 100 | Pending |

### Description

The linked `transfer()`/`transferFrom()` invocations do not check the return value of the function call, which should yield a `true` result in the case of a proper ERC-20 implementation.

```
54          eurxb.transferFrom(msgSender, address(this), _amount);
```

```
87          eurxb.transfer(msgSender, _amount.add(interest));
```

```
100         eurxb.transfer(msgSender, interest);
```

### Recommendation

It is recommended to use SafeERC20 or make sure that the value returned from `transfer()` and `transferFrom()` is checked.

# CCS-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/strategies/CvxCrvStrategy.sol (79bf1fe): 33, 43, 54 | Pending |

## Description

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

- `CvxCrvStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and various information for the `Cvx` pool;
- `CvxCrvStrategy.setPoolIndex()`, which decides which `Cvx` pool the strategy will be using;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `CvxCrvStrategy.deposit()`, which deposits the `_want` tokens in the contract to the associated `Cvx` pool;
- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt how the strategy operates.

## Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# CCS-02 | Inconsistent Logic for `_want` Token

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Major | projects/XBE-finance/contracts/main/strategies/CvxCrvStrategy.sol (79bf1fe): 119~120, 141~143 | Pending |

## Description

The function `convertTokens()` converts a user's `crv` tokens into `CvxCrv` tokens and stores them in the contract.

```
117     function convertTokens(uint256 _amount) external {
118         _convertTokens(_amount);
119         IERC20 _stakingToken = IERC20(_want);
120         uint256 cvxCrvAmount = _stakingToken.balanceOf(address(this));
121         _stakingToken.safeTransfer(msg.sender, cvxCrvAmount);
122     }
```

Following this logic, `_stakingToken` and `_want` both indicate the `CvxCrv` token.

However, the function `convertAndStakeTokens()` suggests that `_stakingToken` and `_want` are the `CvxCrv` LP token.

```
135         uint256 actualCurveCvxCrvLPAmount = stableSwapPool.add_liquidity(
136             _amounts,
137             minCurveCvxCrvLPAmount,
138             address(this)
139         );
140 
141         IERC20 _stakingToken = IERC20(_want);
142         address vault = IController(controller).vaults(_want);
143         _stakingToken.approve(vault, actualCurveCvxCrvLPAmount);
```

There is an inconsistency with regards to the definition of the `IERC20(_want)` token.

In the deployed CvxCrvStrategy contract, the `_want` token is `cvxcrv-f`.

The function `_convertTokens()` deposits `CRV` tokens to `crvdepositor`, which mints for the sender `cvxCRV` tokens, **not** `cvxcrv-f`.

## Recommendation

We recommend changing the `convertTokens()` function to transfer `cvxCRV` tokens instead of the `_want` token.

## CCS-03 | Movement of Funds After Changing `poolIndex`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/main/strategies/CvxCrvStrategy.sol (79bf1fe): 44 | Pending |

## Description

The funds in the pool corresponding to `poolSettings.poolIndex` are not transferred out when changing `poolSettings.poolIndex`. This may cause unexpected errors when calling `_withdrawSome()`.

```
89          require(
90              IBooster(poolSettings.convexBooster).withdraw(
91                  poolSettings.poolIndex,
92                  _amount
93              ),
94              "!withdrawSome"
95          );
```

## Recommendation

We recommend implementing certain logic to avoid insufficient funds or unexpected errors when withdrawing funds associated with `poolSettings.poolIndex`.

## CCS-04 | Unhandled Return Values

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/main/strategies/CvxCrvStrategy.sol (79bf1fe): 66, 87 | Pending |

### Description

The return values of the following function calls are not properly handled:

```
66              IBooster(poolSettings.convexBooster).depositAll(
67                  poolSettings.poolIndex,
68                  true
69              );
```

```
87          IRewards(poolSettings.crvRewards).withdraw(_amount, true);
```

Ignoring the return value of these functions may cause unexpected exceptions.

### Recommendation

We recommend checking the return values of the aforementioned functions and handling both success and failure cases based on the business logic.

## CCS-05 | Potential Sandwich Attacks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/strategies/CvxCrvStrategy.sol (79bf1fe): 135 | Pending |

### Description

A sandwich attack might occur when an attacker observes a transaction that swaps tokens or adds liquidity without setting sufficient restrictions on slippage or a minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction is attacked) a transaction to sell the asset.

The following function is vulnerable to a sandwich attack, especially when the input amount is large:

- `convertAndStakeTokens()`

```
135         uint256 actualCurveCvxCrvLPAmount = stableSwapPool.add_liquidity(
136             _amounts,
137             minCurveCvxCrvLPAmount,
138             address(this)
139         );
```

Although there is a `minCurveCvxCrvLPAmount` parameter, if this argument is sufficiently low, a sandwich attack can be viable.

### Recommendation

We recommend setting minimum amounts based on prices that cannot be manipulated, such as a time-weighted average price that is provided by an oracle.

# CCS-06 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/strategies/CvxCrvStrategy.sol (79bf1fe): 43 | Pending |

## Description

The following function affects the status of sensitive state variables and should be able to emit events as notifications:

- `setPoolIndex()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned function.

## CVX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/strategies/CVXStrategy.sol (79bf1fe): 13, 24 | Pending |

### Description

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

- `CVXStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and the `cvxRewards`;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `CVXStrategy.deposit()`, which stakes the `_want` tokens in the contract to `cvxRewards`;
- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt how the strategy operates.

### Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## CXB-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/Controller.sol (79bf1fe): 63, 76, 86, 96, 101, 121, 127, 140, 154, 165, 185 | Pending |

## Description

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

- `Controller.configure()`, which sets the addresses for the `_treasury`, the `strategist`, and transfers ownership to `_governance`;
- `Controller.inCaseTokensGetStuck()`, which transfers tokens in the contract to `_owner`;
- `Controller.inCaseStrategyTokenGetStuck()`, which transfers non-want tokens in a strategy contract to this contract;
- `Controller.setWithdrawAbility()`, which decides which addresses can withdraw a strategy's tokens to its associated vault;
- `Controller.setTreasury()`, which decides the address for `_treasury`;
- `Controller.setStrategist()`, which decides the address for `strategist`;
- `Controller.setVault()`, which decides the vault address associated to a token;
- `Controller.setConverter()`, which decides the address for a token convertor;
- `Controller.setStrategy()`, which sets an approved strategy for a token;
- `Controller.setApprovedStrategy()`, which decides if a strategy is approved or not for a token;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `strategist` has the authority over the following functions:

- `Controller.inCaseStrategyTokenGetStuck()`, which transfers non-want tokens in a strategy contract to this contract;
- `Controller.setVault()`, which decides the vault address associated to a token;
- `Controller.setConverter()`, which decides the address for a token convertor;
- `Controller.setStrategy()`, which sets an approved strategy for a token.

Addresses with `canWithdraw[address] = true` also have authority over the following function:

- `Controller.withdraw()`, which transfers a strategy's `_want` tokens to its associated vault.

Any compromise to a privileged account may allow the hacker to take advantage of this and jeopardize the operations of the project.

## Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# CXB-02 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/Controller.sol (79bf1fe): 76, 86, 101, 121, 127, 140, 154, 165, 185 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `inCaseTokensGetStuck()`
- `inCaseStrategyTokenGetStuck()`
- `setWithdrawAbility()`
- `setTreasury()`
- `setStrategist()`
- `setVault()`
- `setConverter()`
- `setStrategy()`
- `setApprovedStrategy()`

## Recommendation

We recommend adding events for sensitive actions and emit them in the aforementioned functions.

## FTT-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/FeeToTreasuryTransporter.sol (79bf1fe): 41, 45, 57, 61, 65, 69, 75 | Pending |

### Description

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

- `FeeToTreasuryTransporter.addTokenToConvert()`, which includes a token to be converted into reward tokens;
- `FeeToTreasuryTransporter.removeTokenToConvert()`, which removes a token to be converted into rewards tokens;
- `FeeToTreasuryTransporter.setRewardsToken()`, which decides the address for the reward token;
- `FeeToTreasuryTransporter.setTreasury()`, which sets the address for the treasury;
- `FeeToTreasuryTransporter.setUniswapRouter()`, which sets the address for the Uniswap contract;
- `FeeToTreasuryTransporter.sendRewardToTreasure()`, which transfers reward tokens in the contract to the treasury;
- `FeeToTreasuryTransporter.convertToRewardsToken()`, which swaps tokens in the contract for reward tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

Any compromise to the `_owner` account may allow the hacker to take advantage of this and disrupt the collection of fees.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# FTT-02 | Potential Sandwich Attacks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/FeeToTreasuryTransporter.sol (79bf1fe): 107 | Pending |

## Description

A sandwich attack might occur when an attacker observes a transaction that swaps tokens or adds liquidity without setting sufficient restrictions on slippage or a minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction is attacked) a transaction to sell the asset.

The following function is vulnerable to a sandwich attack, especially when the input amount is large:

- `convertToRewardsToken()`

```
107             actualAmounts[i] = uniswapRouter.swapExactTokensForTokens(
108                 token.balanceOf(address(this)),
109                 amountsOutMin[i],
110                 path,
111                 treasury,
112                 deadlines[i]
113             )[1];
```

Although there is a `amountsOutMin[i]` parameter, if this argument is sufficiently low, a sandwich attack can be viable.

## Recommendation

We recommend setting minimum amounts based on prices that cannot be manipulated, such as a time-weighted average price that is provided by an oracle.

## FTT-03 | Unchecked Value of ERC-20 `transfer()` Call

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/main/FeeToTreasuryTransporter.sol (79bf1fe): 72 | Pending |

### Description

The linked `transfer()` invocation does not check the return value of the function call, which should yield a `true` result in the case of a proper ERC-20 implementation.

```
72          token.transfer(treasury, balance);
```

### Recommendation

It is recommended to use SafeERC20 or make sure that the value returned from `transfer()` and 'transferFrom()' is checked.

# FTT-04 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/FeeToTreasuryTransporter.sol (79bf1fe): 41, 45, 57, 61, 65, 69 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `addTokenToConvert()`
- `removeTokenToConvert()`
- `setRewardsToken()`
- `setTreasury()`
- `setUniswapRouter()`
- `sendRewardToTreasure()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

## HSX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/strategies/HiveStrategy.sol (79bf1fe): 23, 33, 44 | Pending |

### Description

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

- `HiveStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and various information for the `Cvx` pool;
- `HiveStrategy.setPoolIndex()`, which decides which `Cvx` pool the strategy will be using;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `HiveStrategy.deposit()`, which deposits the `_want` tokens in the contract to the associated `Cvx` pool;
- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt how the strategy operates.

### Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## HSX-02 | Movement of Funds After Changing `poolIndex`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/main/strategies/HiveStrategy.sol (79bf1fe): 34 | Pending |

### Description

The funds in the pool corresponding to `poolSettings.poolIndex` are not transferred out when changing `poolSettings.poolIndex`. This may cause unexpected errors when calling `_withdrawSome()`.

```
80          require(
81              IBooster(poolSettings.convexBooster).withdraw(
82                  poolSettings.poolIndex,
83                  _amount
84              ),
85              "!withdrawSome"
86          );
```

### Recommendation

We recommend implementing certain logic to avoid insufficient funds or unexpected errors when withdrawing funds associated with `poolSettings.poolIndex`.

## HSX-03 | Unhandled Return Values

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/main/strategies/HiveStrategy.sol (79bf1fe): 57, 78 | Pending |

### Description

The return values of the following function calls are not properly handled:

```
57              IBooster(poolSettings.convexBooster).depositAll(
58                  poolSettings.poolIndex,
59                  true
60              );
```

```
78          IRewards(poolSettings.crvRewards).withdraw(_amount, true);
```

Ignoring the return value of these functions may cause unexpected exceptions.

### Recommendation

We recommend checking the return values of the aforementioned functions and handling both success and failure cases based on the business logic.

## HSX-04 | Unused Variable

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/strategies/HiveStrategy.sol (79bf1fe): 47 | Pending |

### Description

In the function `deposit()`, the local variable `_amount` is never used.

```
47              uint256 _amount = wantToken.balanceOf(address(this));
```

### Recommendation

We recommend removing the unused variable or implementing a use case for the variable.

## HSX-05 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/strategies/HiveStrategy.sol (79bf1fe): 33 | Pending |

### Description

The following function affects the status of sensitive state variables and should be able to emit events as notifications:

- `setPoolIndex()`

### Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned function.

## LSX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/LockSubscription.sol (79bf1fe): 31, 36, 41, 45, 49, 53 | Pending |

### Description

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

- `LockSubscription.setEventSource()`, which sets the address for `eventSource`;
- `LockSubscription.addSubscriber()`, which adds an address to the set of subscribers;
- `LockSubscription.removeSubscriber()`, which removes an address from the set of subscribers;
- `LockSubscription.pause()`, which prevents `processLockEvent()` from being called;
- `LockSubscription.unpause()`, which re-enables the `processLockEvent()` function;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `eventSource` has the authority over the following function:

- `LockSubscription.processLockEvent()`, which processes a lock event for an account at each subscriber.

Any compromise to the `_owner` or `eventSource` account may allow the hacker to take advantage of this and allow a malicious subscriber to be called.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` and `eventSource` accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## LSX-02 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/LockSubscription.sol (79bf1fe): 31, 36, 41, 53 | Pending |

### Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setEventSource()`
- `addSubscriber()`
- `removeSubscriber()`
- `processLockEvent()`

### Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

## LSX-03 | Function Visibility Optimization

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/LockSubscription.sol (79bf1fe): 23, 27, 31 | Pending |

### Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope:

- `subscribersCount()`
- `subscriberAt()`
- `setEventSource()`

The functions that are never called internally within the contract should have external visibility.

### Recommendation

We advise that the functions' visibility specifiers are set to `external`, optimizing the gas cost of the functions.

# RDR-01 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/staking_rewards/RewardsDistributionRecipient.sol (79bf1fe): 20 | Pending |

## Description

The following function affects the status of sensitive state variables and should be able to emit events as notifications:

- `setRewardsDistribution()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned function.

# RPX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/ReferralProgram.sol (79bf1fe): 78, 97, 150, 163 | Pending |

## Description

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

- `ReferralProgram.changeDistribution()`, which changes how fees are distributed among a user's chain of referrals;
- `ReferralProgram.addNewToken()`, which adds a new token to be considered for collecting fees;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, any `controller` for a registered vault has the authority over the following functions:

- `ReferralProgram.registerUser()`, which registers a user and sets who their referral is;
- `ReferralProgram.feeReceiving()`, which distributes a fee among the chain of referrals for a user.

Any compromise to the `_owner` or a `controller` account may allow the hacker to take advantage of this and jeopardize how fees are distributed.

## Recommendation

We advise the XBE Finance team to carefully manage the `_owner` and all `controller` accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## RPX-02 | Unable to Remove Tokens

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/ReferralProgram.sol (79bf1fe): 122 | Pending |

### Description

In the function `claimRewardsFor()`, there is a `for` loop that runs through the list of supported tokens. If there are too many supported tokens, transactions may become very expensive.

In addition, the contract has an `addNewToken()` function to add supported tokens, but no method to remove depreciated or unwanted tokens.

### Recommendation

We recommend adding a function to remove supported tokens. It is important that this function also sends out all available rewards before removal.

## RXB-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/Registry.sol (79bf1fe): 45, 53, 63, 91 | Pending |

### Description

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

- `Registry.addVault()`, which adds a vault and its controller to the registry;
- `Registry.addWrappedVault()`, which adds a wrapped vault, its controller, and its unwrapped vault to the registry;
- `Registry.addDelegatedVault()`, which adds a delegated vault and its controller to the registry;
- `Registry.removeVault()`, which removes a vault from the registry;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

Any compromise to the `_owner` account may allow the hacker to take advantage of this and manipulate the information for vaults this project deals with.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## SRX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/staking_rewards/StakingRewards.sol (79bf1fe): 164, 168, 174 | Pending |

### Description

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

- `StakingRewards.pause()`, which prevents users from staking;
- `StakingRewards.unpause()`, which allows users to stake again;
- `RewardsDistributionRecipient.setRewardsDistribution()`, which changes the address for `rewardsDistribution`;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `rewardsDistribution` has the authority over the following function:

- `StakingRewards.notifyRewardAmount()`, which updates the reward available.

Any compromise to the `_owner` or `rewardsDistribution` account may allow the hacker to take advantage of this and disrupt how rewards are distributed.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` and `rewardsDistribution` accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## SRX-02 | Possible for Rewards to Be Stuck if There Are No Stakers

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/staking_rewards/StakingRewards.sol (79bf1fe): 81 | Pending |

### Description

The amount of rewards that can be distributed is based on `rewardPerTokenStored`, which can only increase if `totalSupply > 0`.

Hence if there are no stakers, then `rewardPerTokenStored` does not increase, but `lastUpdateTime` can increase, such as when `notifyRewardAmount()` or `updateReward()` is called.

If `T` represents the duration of when `totalSupply = 0`, then `rewardRate * T` amount of rewards will be forever lost.

### Recommendation

We recommend implementing a way for these lost rewards to be added to `rewardPerTokenStored`.

## SRX-03 | Potential Error When `stakingToken` Is the Same As `rewardsToken`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/staking_rewards/StakingRewards.sol (79bf1fe): 193 | Pending |

### Description

In the function `notifyRewardAmount()`, there is a check to ensure that the contract's rewards token balance does not exceed the provided reward.

```
193         uint256 balance = rewardsToken.balanceOf(address(this));
194         require(
195             rewardRate <= balance.div(rewardsDuration),
196             "Provided reward too high"
197         );
```

However, it is possible for the staking token to be the same as the rewards token. In this case, `notifyRewardAmount()` may provide a reward that includes users' stakes, meaning that users will be unable to withdraw their stakes if it has already been given out as rewards.

### Recommendation

If the staking token and the reward token are the same, we recommend having the `rewardsDistribution` be a contract that always transfers the correct reward amount when calling `notifyRewardAmount()`.

## SSX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/strategies/SushiStrategy.sol (79bf1fe): 16 | Pending |

### Description

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

- `SushiStrategy.configure()`, which sets the addresses for the `_want` token, the `controller`, the `governance`, and various information for the liquidity pool;
- `BaseStrategy.setController()`, which sets the address for `controller`;
- `BaseStrategy.setWant()`, which decides the `_want` tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `controller` has the authority over the following functions:

- `BaseStrategy.withdraw(address)`, which transfers non-`_want` tokens to `controller`;
- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

The role `IController(controller).vaults(_want)` also has authority over the following functions:

- `BaseStrategy.withdraw(uint256)`, which transfers `_want` tokens to the vault associated with the `_want` token;
- `ClaimableStrategy.claim()`, which transfers a token held by the contract to the vault associated with the `_want` token.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt how the strategy operates.

### Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the

protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## SSX-02 | Unimplemented Functions in `SushiStrategy`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Discussion | projects/XBE-finance/contracts/main/strategies/SushiStrategy.sol (79bf1fe) : 28 | Pending |

## Description

The contract `SushiStrategy` has functions that do not seem to be fully implemented, such as `deposit()` and `getRewards()`. We would like to know if this contract is not yet finished or if it is, the purpose of the contract.

## SVX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/vaults/SushiVault.sol (79bf1fe): 11 | Pending |

### Description

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

- `SushiVault.configure()`, which initializes important information like the staking token, rewards tokens, rewards duration, the controller, and transfers ownership to `_governance`;
- `BaseVaultV2.setTrustworthyEarnCaller()`, which decides the address for `trustworthyEarnCaller`;
- `BaseVaultV2.setController()`, which decides the address for `_controller`;
- `BaseVaultV2.setRewardsDistribution()`, which decides the address for `rewardsDistribution`;
- `BaseVaultV2.pause()`, which prevents deposits;
- `BaseVaultV2.unpause()`, which re-enables deposits;
- `BaseVaultV2.setRewardsDuration()`, which decides the amount of time for a reward to be paid out;
- `BaseVaultV2.addRewardToken()`, which adds a token that can be given as rewards;
- `BaseVaultV2.removeRewardToken()`, which removes a token that can be given as rewards;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `rewardsDistribution` has the authority over the following function:

- `BaseVaultV2.notifyRewardAmount()`, which adds a reward to be distributed.

The role `trustworthyEarnCaller` also has the authority over the following function:

- `BaseVaultV2.earn()`, which transfers staking tokens to the controller and claims any reward tokens.

Any compromise to the `_owner`, `rewardsDistribution`, or `trustworthyEarnCaller` account may allow the hacker to take advantage of this and disrupt how rewards are distributed.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner`, `rewardsDistribution`, and `trustworthyEarnCaller` accounts' private keys to avoid any potential risks of being hacked. In general,

we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## SXB-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| **Centralization / Privilege** | **Major** | projects/XBE-finance/contracts/main/SimpleXBEInflation.sol (79bf1fe): 36, 57, 70 | Pending |

### Description

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

- `SimpleXBEInflation.configure()`, which decides the token to mint, the maximum amount to mint, and the amount to mint in each period;
- `SimpleXBEInflation.setXBEReceiver()`, which allows an address to receive minted tokens and decides how much to receive;
- `SimpleXBEInflation.removeXBEReceiver()`, which decides which address will no longer receive minted tokens;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

Any compromise to the `_owner` account may allow the hacker to take advantage of this and disrupt how minted tokens are distributed.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## SXB-02 | Incorrect Weight When Adding a Removed `_xbeReceiver`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/main/SimpleXBEInflation.sol (79bf1fe): 70 | Pending |

### Description

When `setXBEReceiver(_xbeReceiver, _weight)` is called, the `sumWeight` value decreases by `weights[_xbeReceiver]` and increases by `_weight`. The value of `weights[_xbeReceiver]` is also updated to `_weight`.

However, when `removeXBEReceiver(_xbeReceiver)` is called, the `sumWeight` value decreases by `weights[_xbeReceiver]` but `weights[_xbeReceiver]` remains unchanged.

Hence when `setXBEReceiver()` is called on a removed receiver, the `sumWeight` value is decreased by `weights[_xbeReceiver]` more than it should be.

For example, if we currently have `sumWeight = 100` and `weights[receiver] = 50`, then when `removeXBEReceiver(receiver)` is called, we have `sumWeight = 50` and `weights[receiver] = 50`. If then `setXBEReceiver(receiver, 50)` is called, we would have `sumWeight = 50` and `weights[receiver] = 50`. Thus `sumWeight` decreased by 50 when all receivers' weights did not change.

### Recommendation

We recommend changing a receiver's weight to 0 when they are removed from the `xbeReceivers` set.

# SXB-03 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/SimpleXBEInflation.sol (79bf1fe): 57, 70, 92 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setXBEReceiver()`
- `removeXBEReceiver()`
- `mintForContracts()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

## TWX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/TokenWrapper.sol (79bf1fe): 35, 44 | Pending |

## Description

In the contract `TokenWrapper`, the role `MINTER_ROLE` has the authority over the following functions:

- `TokenWrapper.mint(uint256)`, which transfers `wrappedtoken` tokens from `MINTER_ROLE` to the contract and mints the same amount of tokens to `MINTER_ROLE`;
- `TokenWrapper.burn()`, which burns an amount of tokens for `MINTER_ROLE` and transfers the same amount of `wrappedtoken` tokens to `MINTER_ROLE`;
- `ERC20PresetMinterPauser.mint(address,uint256)`, which mints an amount of tokens to an address.
- `AccessControl.renounceRole()`, which removes the caller's `MINTER_ROLE` authority.

In addition, the role `PAUSER_ROLE` has the authority over the following functions:

- `ERC20PresetMinterPauser.pause()`, which disables token transfers, mints, and burns;
- `ERC20PresetMinterPauser.unpause()`, which re-enables token transfers, mints, and burns.
- `AccessControl.renounceRole()`, which removes the caller's `PAUSER_ROLE` authority.

In addition, the role `DEFAULT_ADMIN_ROLE` has the authority over the following functions:

- `AccessControl.grantRole()`, which grants a role to an address;
- `AccessControl.revokeRole()`, which revokes a role from an address;
- `AccessControl.renounceRole()`, which removes the caller's `DEFAULT_ADMIN_ROLE` authority.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt how the token operates.

## Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# TWX-02 | Function Visibility Optimization

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/TokenWrapper.sol (79bf1fe): 35 , 44 | Pending |

## Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope:

- `mint()`
- `burn()`

The functions that are never called internally within the contract should have external visibility.

## Recommendation

We advise that the functions' visibility specifiers are set to `external`, optimizing the gas cost of the functions.

## TXB-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/Treasury.sol (79bf1fe): 46, 64, 68, 76, 82, 86, 90, 102, 128 | Pending |

### Description

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

- `Treasury.configure()`, which decides the information for rewards, for swapping tokens, and also transfers ownership to `_governance`;
- `Treasury.setRewardsToken()`, which decides the address of the rewards token;
- `Treasury.setSlippageTolerance()`, which decides the amount of slippage in token swaps;
- `Treasury.setRewardsDistributionRecipientContract()`, which decides the address of the contract that distributes rewards;
- `Treasury.setAuthorized()`, which decides which addresses are authorized to convert tokens;
- `Treasury.addTokenToConvert()`, which adds a token to be converted to rewards tokens;
- `Treasury.removeTokenToConvert()`, which removes a token to be converted;
- `Treasury.toGovernance()`, which transfers any token in the contract to `_owner`;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, addresses that are authorized have access to the following function:

- `Treasury.convertToRewardsToken()`, which swaps a token to the rewards token via Uniswap.

Any compromise to the `_owner` or any authorized account may allow the hacker to take advantage of this and disrupt how the rewards program works.

### Recommendation

We advise the XBE Finance team to carefully manage the `_owner` and all authorized accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# TXB-02 | Unhandled Return Values

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | projects/XBE-finance/contracts/main/Treasury.sol (79bf1fe): 118~124 | Pending |

## Description

The return values of the following function call is not properly handled:

```
118         uniswapRouter.swapExactTokensForTokens(
119             amount,
120             amountOutMin,
121             path,
122             address(this),
123             block.timestamp + swapDeadline
124         );
```

Ignoring the return value of this function may cause unexpected exceptions.

## Recommendation

We recommend checking the return values of the aforementioned function and handling both success and failure cases based on the business logic.

## TXB-03 | Potential Sandwich Attacks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/Treasury.sol (79bf1fe): 118 | Pending |

### Description

A sandwich attack might occur when an attacker observes a transaction that swaps tokens or adds liquidity without setting sufficient restrictions on slippage or a minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction is attacked) a transaction to sell the asset.

The following function is vulnerable to sandwich attacks, especially when the input amount is large:

- `convertToRewardsToken()`

```
118         uniswapRouter.swapExactTokensForTokens(
119             amount,
120             amountOutMin,
121             path,
122             address(this),
123             block.timestamp + swapDeadline
124         );
```

Although there is a `amountOutMin` parameter, if this argument is sufficiently low, a sandwich attack can be viable.

### Recommendation

We recommend setting minimum amounts based on prices that cannot be manipulated, such as a time-weighted average price that is provided by an oracle.

# TXB-04 | Slippage Tolerance is Not Used

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Informational | projects/XBE-finance/contracts/main/Treasury.sol (79bf1fe): 34 | Pending |

## Description

The contract 'Treasury' has a `slippageTolerance` variable to decide the maximum amount of price movement allowed in token swaps.

This contract only performs swaps in the function `convertToRewardsToken()`, but `slippageTolerance` is not used in this swap.

## Recommendation

We recommend removing the variable `slippageTolerance` and the function `setSlippageTolerance()` if a slippage tolerance is not intended to be used. Otherwise, we recommend incorporating a slippage tolerance in token swaps.

# TXB-05 | Rewards Are Not Updated Prior to Changing Rewards Token

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/main/Treasury.sol (79bf1fe): 64~66 | Pending |

## Description

The function `setRewardsToken()` changes the token to be used as rewards for voters.

However, if the rewards token is changed while there the contract still contains rewards tokens, then voters will miss out and be unable to claim these rewards.

## Recommendation

We recommend calling `toVoters()` before changing the rewards token so that all rewards will be distributed.

# TXB-06 | Function Visibility Optimization

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/Treasury.sol (79bf1fe): 102 | Pending |

## Description

The following function is declared as `public` and is not invoked in any of the contracts contained within the project's scope:

- `convertToRewardsToken()`

The function that is never called internally within the contract should have external visibility.

## Recommendation

We advise that the function's visibility specifiers are set to `external`, optimizing the gas cost of the function.

# VPX-01 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/governance/utils/VotingPausable.sol (79bf1fe): 16, 21 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setPaused()`
- `transferOwnership()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

## VSR-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 127, 134, 146, 151, 158, 165, 198 | Pending |

### Description

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

- `VotingStakingRewards.setRewardsDistribution()`, which changes the address of `rewardsDistribution`;
- `VotingStakingRewards.setInverseMaxBoostCoefficient()`, which changes the minimum boost level;
- `VotingStakingRewards.setPenaltyPct()`, which changes the amount of penalty applied to premature bonded withdrawals;
- `VotingStakingRewards.setBondedLockDuration()`, which changes how long bonded stakes are locked up for;
- `VotingStakingRewards.setBoostLogicProvider()`, which changes the address of `boostLogicProvider`;
- `VotingStakingRewards.setAddressWhoCanAutoStake()`, which decides whether an address can stake for others or not;
- `VotingOwnable.renounceOwnership()`, which disables all functions that can only be called by `_owner`;
- `VotingOwnable.transferOwnership()`, which changes the address of `_owner`.

In addition, the role `rewardsDistribution` has authority over the following function:

- `VotingStakingRewards.notifyRewardAmount()`, which updates the rewards to be provided.

The role `pauser` has authority over the following functions:

- `VotingPausable.setPaused()`, which can disable or re-enable the functions `stakeFor()` and `stake()`;
- `VotingPausable.transferOwnership()`, which changes the address of `pauser`.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt staking and reward mechanisms.

## Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# VSR-02 | Calculation of Rewards Uses Current Boost Level

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Major | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe) : 397 | Pending |

## Description

The calculation of a user's rewards uses the user's current boost level. This means that if a user creates a lock for 100 weeks in `veXBE` to obtain the maximum boost level and `updateReward(user)` is only called after these 100 weeks, then `veXBE.balanceOf(user) = 0` so the user will actually receive rewards based on the minimum boost level and lose out on rewards.

The only way for a user to obtain the correct amount of rewards is to have `updateReward(user)` be called on a continuous basis.

Another consequence of rewards being based on a user's current boost level is that it is possible to always obtain the maximum amount of rewards. Before a user calls a function with the modifier `updateReward()`, they can create a lock or increase the lock time in `veXBE` so that they have the maximum boost level. By doing so, their rewards will always be calculated using the maximum boost level, even though they do not have the maximum boost between `updateReward()` calls.

## Recommendation

We recommend revisiting the logic of how a user's boost level is applied to rewards.

# VSR-03 | Inaccurate Boost Level Calculation

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Medium | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 330 | Pending |

## Description

The document regarding boost levels states that the boost level is proportional to a user's lockup time, with each week remaining of lockup corresponding to 1% of the max boost less the minimum boost.

The calculation of the boost level is given below:

```
330         uint256 res = PCT_BASE
331             .mul(
332                 inverseMaxBoostCoefficient.add(
333                     uint256(100)
334                         .sub(inverseMaxBoostCoefficient)
335                         .mul(veXBE.lockedSupply())
336                         .mul(votingBalance)
337                         .div(votingTotal)
338                         .div(lockedAmount)
339                 )
340             )
341             .div(100);
342
343         return res < MAX_BOOST_LEVEL ? res : MAX_BOOST_LEVEL;
```

The proportional decrease in boost level as the user's lockup time nears is accounted for due to the `votingBalance / votingTotal` value since the user's balance decreases as the lockup time approaches.

However, the value `veXBE.lockedSupply() / lockedAmount` will in general be greater than 1, allowing users to have a higher boost level than they should.

## Recommendation

We recommend only basing a user's boost level on the user's deposits and remaining lockup time.

# VSR-04 | Staking Tokens Are Transferred to Treasury Instead of Reward Tokens

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 109 | Pending |

## Description

In the modifier `updateReward()`, a user's rewards are updated and a percentage of these rewards is sent to the treasury.

However, the amount sent to the treasury is in terms of staking tokens instead of reward tokens.

```
107             if (toTreasury > 0) {
108                 require(
109                     stakingToken.transfer(treasury, toTreasury),
110                     "!boostDelta"
111                 );
```

In the deployed `VotingStakingRewards` contract, the staking and reward tokens are the same so this does not cause issues, but it may in future implementations.

## Recommendation

We recommend transferring reward tokens to the treasury instead of staking tokens.

# VSR-05 | Bypassing Bonded Unlock Times

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 290 | Pending |

## Description

When the function `stakeFor()` is called for a user, a `bondedReward` is created for the user. This prevents the user from withdrawing the staked amount for `bondedLockDuration` unless they are willing to suffer a penalty.

However, the user can call `VeXBE.createLock()` to create a lock for the same amount as the staked amount and then call `withdrawBondedOrWithPenalty()`.

In the execution of `withdrawBondedOrWithPenalty()`, `escrowed` will equal `_balances[user]` so the amount to be withdrawn will be 0, meaning no penalty is applied. At the end of the function call, `bondedRewardLocks[user]` is deleted, meaning the user no longer has a lockup time on the staked amount.

If the lockup duration in `VeXBE` is lower than `bondedLockDuration`, then a user can perform the above to be able to withdraw the staked amount earlier than expected and suffer no penalties.

The current `bondedLockDuration` in the deployed `VotingStakingRewards` is half a day while the minimum lock duration in `VeXBE` is a week. However, since these values can be changed, it may be an issue in the future.

## Recommendation

We recommend a user's `bondedReward.amount` to be decreased by the amount withdrawn in `withdrawBondedOrWithPenalty()` instead of deleting the user's `bondedReward`.

## VSR-06 | Potential Error When `stakingToken` Is the Same As `rewardsToken`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 215 | Pending |

### Description

In the function `notifyRewardAmount()`, there is a check to ensure that the contract's rewards token balance does not exceed the provided reward.

```
215         uint256 balance = rewardsToken.balanceOf(address(this));
216         require(
217             rewardRate <= balance.div(rewardsDuration),
218             "Provided reward too high"
219         );
```

However, in the deployed `VotingStakingRewards` contract, the staking token is the same as the rewards token. This allows the possibility for `notifyRewardAmount()` to provide a reward that includes users' stakes, meaning that users will be unable to withdraw their stakes if it has already been given out as rewards.

The current `rewardsDistribution` is the `Treasury` contract which transfers the necessary amount of rewards to `VotingStakingRewards` when calling `notifyRewardAmount()`, so this issue will not occur.

However, since `rewardsDistribution` can be changed via `setRewardsDistribution()`, it is possible for the above issue to happen.

### Recommendation

We recommend to always have the `rewardsDistribution` be a contract that transfers the correct reward amount when calling `notifyRewardAmount()`.

# VSR-07 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 127, 134, 146, 151, 158, 165 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setRewardsDistribution()`
- `setInverseMaxBoostCoefficient()`
- `setPenaltyPct()`
- `setBondedLockDuration()`
- `setBoostLogicProvider()`
- `setAddressWhoCanAutoStake()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

# VSR-08 | Possible for Rewards to Be Stuck if There Are No Stakers

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Informational | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 186 | Pending |

## Description

The amount of rewards that can be distributed is based on `rewardPerTokenStored`, which can only increase if `totalSupply > 0`.

Hence, if there are no stakers, then `rewardPerTokenStored` does not increase, but `lastUpdateTime` can increase, such as when `notifyRewardAmount()` or `updateReward()` is called.

If `T` represents the duration of when `totalSupply = 0`, then `rewardRate * T` amount of rewards will be forever lost.

## Recommendation

We recommend implementing a way for these lost rewards to be added to `rewardPerTokenStored`.

## VSR-09 | No Restriction on `bondedLockDuration`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Informational | projects/XBE-finance/contracts/governance/VotingStakingRewards.sol (79bf1fe): 155 | Pending |

### Description

The function `setBondedLockDuration()` changes the value of `bondedLockDuration`, which decides the lockup time for certain stakes.

During the initialization of this contract, `bondedLockDuration` is required to be positive, but there is no restriction in the function `setBondedLockDuration()`.

### Recommendation

We recommend adding a restriction to ensure that `bondedLockDuration` is positive.

# VWF-01 | Fee Distribution Inconsistent With Documents

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Inconsistency | Medium | projects/XBE-finance/contracts/main/vaults/base/VaultWithFees.sol (79bf1fe): 136 | Pending |

## Description

In the function `_getFeesOnClaimForToken()`, fees are applied to claim amounts. The first fee is applied to the entirety of `_amount` while the second fee is applied to `_amount` less the first fees and so on for later fee receivers.

```
130         for (uint256 i = 0; i < claimFee.length; i++) {
131             ClaimFee memory _claimFee = claimFee[i];
132             fee = uint256(_claimFee.percentage).mul(_amount).div(PCT_BASE);
133             if (claimFee[i].tokens[_rewardToken]) {
134                 IERC20(_rewardToken).safeTransfer(_claimFee.to, fee);
135             }
136             _amount = _amount.sub(fee);
```

This is inconsistent with the document on the fee breakdown. In this document, it states that a fee of 21% is applied to claimed rewards with a breakdown of 10%/10%/1%, so we should have 3 fee receivers.

The current implementation would have a fee of approximately 19.81% (100 - 90 * 90 * 99). Depending on the order of fee receivers, this may mean a smaller amount is given as revenue for XBE stakers or for referral rewards.

## Recommendation

We recommend to instead apply each fee to the entirety of `_amount`.

## VWF-02 | Possible Integer Overflow

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/main/vaults/base/VaultWithFees.sol (79bf1fe): 57 | Pending |

### Description

In the function `addClaimFeeReceiver()`, there is a requirement that the fee percentage given to a receiver does not cause `sumClaimFee` to exceed `PCT_BASE`.

```
57          require(sumClaimFee + _percentage <= PCT_BASE, "!sumClaimFee overflow");
```

However, as `_percentage` is an arbitrary input, it is possible for `sumClaimFee + _percentage` to overflow while still being under `PCT_BASE`.

### Recommendation

We recommend using `SafeMath` when performing this addition or adding an additional condition to place an upper bound on `_percentage`, such as `_percentage <= PCT_BASE`.

# VWF-03 | `sumClaimFee` Lacks an Upper Bound

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Minor | projects/XBE-finance/contracts/main/vaults/base/VaultWithFees.sol (79bf1fe): 102 | Pending |

## Description

In the function `setClaimFeePercentage()`, a receiver's fee percentage amount is changed and `sumClaimFee` is updated to reflect this change.

However, there are no checks on the value of `sumClaimFee`, allowing it to exceed `PCT_BASE`. This would cause claim fees to be at least 100% of claim amounts.

## Recommendation

We recommend adding a check to ensure that the updated `sumClaimFee` does not exceed `PCT_BASE`.

# VWF-04 | Uninitialized Storage Variable

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Language Specific | Informational | projects/XBE-finance/contracts/main/vaults/base/VaultWithFees.sol (79bf1fe): 58 | Pending |

## Description

The variable `newWeight` in the function `addClaimFeeReceiver()` is an uninitialized storage variable. This means `newWeight` will act as a reference to the first storage slot and can override a critical variable.

For example, if `VaultWithFees` was a contract instead of an abstract contract, `addClaimFeeReceiver()` will overwrite the `_owner` variable. In the `Vault` contract, the first storage slot is the mapping `_balances` from `ERC20Vault`. Since storage values for mappings are not used, issues do not arise.

## Recommendation

We recommend making sure that the first storage slot is inconsequential when inheriting `VaultWithFees`.

# VWF-05 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/vaults/base/VaultWithFees.sol (79bf1fe): 47, 51, 89, 96 | Pending |

## Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `setFeesEnabled()`
- `addClaimFeeReceiver()`
- `removeClaimFeeReceiver()`
- `setClaimFeePercentage()`

## Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

# VXB-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/governance/Voting.sol (79bf1fe): 132, 153, 175, 191, 249 | Pending |

## Description

In the contract `Voting`, the role `MODIFY_SUPPORT_ROLE` has the authority over the following function:

- `Voting.changeSupportRequiredPct()`, which changes the amount of `yea` votes versus `nay` votes a proposal needs to pass.

In addition, the role `MODIFY_QUORUM_ROLE` has authority over the following function:

- `Voting.changeMinAcceptQuorumPct()`, which changes the amount of `yea` votes versus total possible votes a proposal needs to pass.

Also, the role `CREATE_VOTES_ROLE` has authority over the following functions:

- `Voting.newVote(bytes,string)`, which creates a new proposal, votes for the caller in support of the proposal, and executes the proposal if it meets the voting requirements;
- `Voting.newVote(bytes,string,bool,bool)`, which creates a new proposal, may vote for the caller, and may try to execute the proposal;
- `Voting.forward()`, which creates a new proposal, votes for the caller in support of the proposal, and executes the proposal if it meets the voting requirements.

Any compromise to a privileged account may allow the hacker to take advantage of this and disrupt the voting process.

## Recommendation

We advise the XBE Finance team to carefully manage all privileged accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# VXB-02 | Potential Reentrancy Attack

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/governance/Voting.sol (79bf1fe): 401 | Pending |

## Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would otherwise not have run after the external call resolved its effects.

For example, if the `VoterState` of the attacker is `Absent`, then re-entering would allow the attacker to vote twice.

## Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin's ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

# VXB-03 | Function Visibility Optimization

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/governance/Voting.sol (79bf1fe): 273, 288, 311, 347 | Pending |

## Description

The following functions are declared as `public` and are not invoked in any of the contracts contained within the project's scope:

- `canExecute()`
- `canVote()`
- `getVote()`
- `getVoterState()`

The functions that are never called internally within the contract should have external visibility.

## Recommendation

We advise that the functions' visibility specifiers are set to `external`, optimizing the gas cost of the functions.

# VXE-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/vaults/Vault.sol (79bf1fe): 21 | Pending |

## Description

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

- `Vault.configure()`, which initializes important information like the staking token, rewards tokens, rewards duration, the controller, and transfers ownership to `_governance`;
- `VaultWithFees.setFeesEnabled()`, which decides if fees are applied to rewards or not;
- `VaultWithFees.addClaimFeeReceiver()`, which adds an address that receives fees and decides the percentage it receives;
- `VaultWithFees.removeClaimFeeReceiver()`, which removes an address that receives fees;
- `VaultWithFees.setClaimFeePercentage()`, which changes the percentage of fees sent to a receiver;
- `VaultWithFees.setDepositFee()`, which changes the percentage of fees applied to deposits;
- `VaultWithFees.setDepositWallet()`, which changes where deposit fees are sent to;
- `BaseVaultV2.setTrustworthyEarnCaller()`, which decides the address for `trustworthyEarnCaller`;
- `BaseVaultV2.setController()`, which decides the address for `_controller`;
- `BaseVaultV2.setRewardsDistribution()`, which decides the address for `rewardsDistribution`;
- `BaseVaultV2.pause()`, which prevents deposits;
- `BaseVaultV2.unpause()`, which re-enables deposits;
- `BaseVaultV2.setRewardsDuration()`, which decides the amount of time for a reward to be paid out;
- `BaseVaultV2.addRewardToken()`, which adds a token that can be given as rewards;
- `BaseVaultV2.removeRewardToken()`, which removes a token that can be given as rewards;
- `Ownable.renounceOwnership()`, which disables all `_owner` privileged functions;
- `Ownable.transferOwnership()`, which transfers the `_owner` role to an address.

In addition, the role `rewardsDistribution` has the authority over the following function:

- `BaseVaultV2.notifyRewardAmount()`, which adds a reward to be distributed.

The role `trustworthyEarnCaller` also has the authority over the following function:

- `BaseVaultV2.earn()`, which transfers staking tokens to the controller and claims any reward tokens.

Any compromise to the `_owner`, `rewardsDistribution`, or `trustworthyEarnCaller` account may allow the hacker to take advantage of this and disrupt how rewards are distributed.

## Recommendation

We advise the XBE Finance team to carefully manage the `_owner`, `rewardsDistribution`, and `trustworthyEarnCaller` accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

## VXX-01 | Centralization Risk

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Major | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 142, 146, 155, 163 | Pending |

### Description

In the contract `VeXBE`, the role `admin` has the authority over the following functions:

- `VeXBE.setMinLockDuration()`, which sets the minimum lock duration;
- `VeXBE.setVoting()`, which sets the `votingStakingRewards` address;
- `VeXBE.commitTransferOwnership()`, which sets the future `admin` candidate address;
- `VeXBE.applyTransferOwnership()`, which sets the future `admin` candidate to be `admin`.

Any compromise to the `admin` account may allow the hacker to take advantage of this and manipulate the voting protocol.

### Recommendation

We advise the XBE Finance team to carefully manage the `admin` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

# VXX-02 | Anyone Can Allow Themselves to Create Locks for Others

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Medium | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 484, 518 | Pending |

## Description

The function `setCreateAllowance()` sets the bool for `createLockAllowance[msg.sender][_sender]` while the function `createLockFor` checks if `createLockAllwance[msg.sender][_for]` is `true`. This means that anyone can create locks for others without their permission.

For example, if user A calls `setCreateAllowance(B, true)`, then `createLockAllwance[A][B] = true`. Then when A calls `createLockFor(B, _value, _unlocktime)`, they are allowed to do so as `createLockAllwance[A][B] = true`.

A consequence of this is that if user B staked tokens in `VotingStakingRewards` but has not created a lock in `VeXBE`, then user A can create a lock in `VeXBE` for user B for an amount equal to B's staked tokens. This prevents user B from withdrawing their staked tokens for the duration of the created lock.

## Recommendation

We recommend switching the order of the addresses for `createLockAllowance` in one of the functions `setCreateAllowance()` or `createLockFor()`.

# VXX-03 | Check Staking Amount Near the Beginning of Function Call

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 430 | Pending |

## Description

In the function `_depositFor()`, a check is made to see if the user's staked amount is at least the user's locked amount.

```
430         require(
431             IERC20(votingStakingRewards).balanceOf(_addr) >=
432                 uint256(_locked.amount),
433             "notEnoughStake"
434         );
```

This check is performed after creating a checkpoint, but it would be more gas efficient in case of reverts if the check is performed before creating a checkpoint.

## Recommendation

We recommend placing this require statement before creating a checkpoint.

# VXX-04 | Zero Voting Power For Small Locked Amount

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Volatile Code | Informational | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 232, 238 | Pending |

## Description

The calculation of the slope for a locked amount is the amount divided by `MAXTIME = 60,480,000`.

```
238                 uNew.slope = int128(uint256(newLocked.amount) / MAXTIME);
```

A consequence of this is that locked amounts less than `MAXTIME` will have zero voting power.

```
239                 uNew.bias =
240                     uNew.slope *
241                     int128(newLocked.end - block.timestamp);
```

## Recommendation

We recommend having a multiplier applied to slopes so that small locked amounts have some voting power.

## VXX-05 | Lack of Event Emissions for Significant Transactions

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Coding Style | Informational | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 137, 146 | Pending |

### Description

The following functions affect the status of sensitive state variables and should be able to emit events as notifications:

- `_setMinLockDuration()`
- `setVoting()`

### Recommendation

Consider adding events for sensitive actions and emit them in the aforementioned functions.

# VXX-06 | Function Visibility Optimization

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 616 | Pending |

## Description

The following function is declared as `public` and is not invoked in any of the contracts contained within the project's scope:

- `balanceOf(address)`

The functions that are never called internally within the contract should have external visibility.

## Recommendation

We advise that the function's visibility specifiers are set to `external`, optimizing the gas cost of the function.

## VXX-07 | `votingStakingRewards` Never Calls `createLockFor`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Discussion | projects/XBE-finance/contracts/main/VeXBE.sol (79bf1fe): 517 | Pending |

### Description

The function `createLockFor()` only allows users that have an allowance or the `VotingStakingRewards` contract to call it.

```
517         if (msg.sender != votingStakingRewards) {
518             require(createLockAllowance[msg.sender][_for], "!allowed");
519         }
```

However, the contract `VotingStakingRewards` never calls `VeXBE.createLockFor()`. Moreover, this function is not in the interface `IVeXBE`.

### Recommendation

If the `VotingStakingRewards` contract should not call `createLockFor()`, then we recommend removing the check `(msg.sender != votingStakingRewards)` for clarity. Otherwise, we recommend implementing a call to this function in the `VotingStakingRewards` contract.

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

### Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

### Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

# Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.