Smart Contract Security Audit Report

---

zkBob

1. Contents

2. General Information

This report contains information about the results of the security audit of the zkBob (hereafter referred to as "Customer") CDP and Pool smart contracts, conducted by Decurity in the period from 04/13/2023 to 05/01/2023.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repositories:

- CDP: https://github.com/zkBob/bob-cdp-contracts/releases/tag/1.0.0-rc0 (commit 7ba4e7bbc92fc8d49486ec36606c1e6d5ceca7ce)
- zkBob: https://github.com/zkBob/zkbob-contracts/releases/tag/1.1.0-rc2 (commit 444bd591bc464b254366525aa23cca5e84ebceb6)

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Relayers.

- Oracles,
- DEX Pools.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered a few potentially high-risk issues, the other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of May 09, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Mint/burn quota bypass in BalancedMinter	zkbob-contracts/src/minters/BalancedMinter.sol	**High**	Fixed
Any token can be transferred from liquidation bot	cdp-nft-liqbot/contracts/src/Bot.sol	**High**	Fixed
Centralization risk	-	**Medium**	Acknowledged
Pool limitations lead to DoS	-	**Medium**	Acknowledged
Direct deposit limits do not match the pool limits	zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol	**Medium**	Acknowledged
Direct deposits DoS	zkbob-contracts/src/zkbob/ZkBobDirectDeposi	**Medium**	Acknowledged

Issue	Contract	Risk Level	Status
	tQueue.sol, zkbob-contracts/src/zkbob/manager/MutableOperatorManager.sol zkbob-contracts/src/zkbob/manager/SimpleOperatorManager.sol		
User controlled calls in collectAndIncreaseAmount	bob-cdp-contracts/src/Vault.sol	Medium	Fixed
Missing return value check of the ERC20 transfers	zkbob-contracts/src/utils/UniswapV3Seller.sol	Medium	Fixed
Wrong parameter passed to event	zkbob-contracts/src/minters/BalancedMinter.sol	Low	Fixed
Missing workflow invariants	-	Low	Fixed
Infinite allowance in _checkAllowance	bob-cdp-contracts/src/Vault.sol	Low	Fixed
No events emitted on state changes	bob-cdp-contracts/src/oracles/UniV3Oracle.sol zkbob-contracts/src/minters/BaseMinter.sol zkbob-contracts/src/minters/FlashMinter.sol zkbob-contracts/src/minters/SurplusMinter.sol	Low	Fixed
Differences in Oracle implementations	bob-cdp-contracts/src/oracles/QuickswapV3Orac	Low	Fixed

Issue	Contract	Risk Level	Status
	le.sol bob-cdp-contracts/src/oracles/UniV3Oracle.sol		
Non-optimal increment/decrement	zkbob-contracts/src/token/ERC20Recovery.sol zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol bob-cdp-contracts/src/Vault.sol bob-cdp-contracts/src/oracles/ChainlinkOracle.sol	Info	Fixed
Unnecessary checked arithmetic in loop	zkbob-contracts/src/token/ERC20Recovery.sol zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol bob-cdp-contracts/src/Vault.sol bob-cdp-contracts/src/oracles/ChainlinkOracle.sol bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol bob-cdp-contracts/src/oracles/UniV3Oracle.sol bob-cdp-contracts/src/token/ERC721/ERC721.sol	Info	Acknowledged

Issue	Contract	Risk Level	Status
Non-optimal conditional statements	zkbob-contracts/src/libraries/ZkAddress.sol zkbob-contracts/src/utils/EIP712.sol zkbob-contracts/src/zkbob/ZkBobPool.sol zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol bob-cdp-contracts/src/Vault.sol bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol bob-cdp-contracts/src/utils/VaultAccessControl.sol	Info	Acknowledged
Unused function parameter	zkbob-contracts/src/minters/BaseMinter.sol zkbob-contracts/src/minters/SurplusMinter.sol	Info	Acknowledged
Cache array length outside the loop	/bob-cdp-contracts/src/Vault.sol /bob-cdp-contracts/src/Vault.sol /bob-cdp-contracts/src/oracles/ChainlinkOracle.sol	Info	Fixed
String error messages used instead of custom errors	bob-cdp-contracts/src/token/ERC721/ERC721.sol bob-cdp-contracts/src/token/ERC721/ERC721Enumerable.sol zkbob-	Info	Acknowledged

Issue	Contract	Risk Level	Status
	contracts/src/BobVault.sol zkbob-contracts/src/minters/BalancedMinter.sol zkbob-contracts/src/minters/BaseMinter.sol zkbob-contracts/src/minters/DebtMinter.sol zkbob-contracts/src/minters/FaucetMinter.sol zkbob-contracts/src/minters/FlashMinter.sol zkbob-contracts/src/minters/SurplusMinter.sol zkbob-contracts/src/proxy/EIP1967Admin.sol zkbob-contracts/src/proxy/EIP1967Proxy.sol zkbob-contracts/src/token/BaseERC20.sol zkbob-contracts/src/token/ERC20Blocklist.sol zkbob-contracts/src/token/ERC20MintBurn.sol zkbob-contracts/src/token/ERC20Permit.sol zkbob-contracts/src/token/ERC20Recovery.sol zkbob-contracts/src/token/ERC677.sol		

Issue	Contract	Risk Level	Status
	zkbob-contracts/src/utils/Claimable.sol zkbob-contracts/src/utils/Ownable.sol zkbob-contracts/src/utils/UniswapV3Seller.sol zkbob-contracts/src/yield/AAVEYieldImplementation.sol zkbob-contracts/src/yield/YieldConnector.sol zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol zkbob-contracts/src/zkbob/ZkBobPool.sol zkbob-contracts/src/zkbob/ZkBobPoolETH.sol zkbob-contracts/src/zkbob/manager/MutableOperatorManager.sol zkbob-contracts/src/zkbob/manager/SimpleOperatorManager.sol zkbob-contracts/src/zkbob/manager/kyc/SimpleKYCProviderManager.sol zkbob-contracts/src/zkbob/utils/KycProvidersManagerStorage.sol zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol		

Issue	Contract	Risk Level	Status
Inconsistent variable naming style	bob-cdp-contracts/src/interfaces/ICDP.sol bob-cdp-contracts/src/Vault.sol	Info	Fixed
Outdated OpenZeppelin library	zkbob-contracts/lib/@openzeppelin	Info	Fixed
Missing zero transfer token amount check in ZkBobPool	zkbob-contracts/src/zkbob/ZkBobPool.sol	Info	Fixed
Unused variables	bob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol	Info	Fixed
Local variable shadowing	zkbob-contracts/src/minters/FlashMinter.sol	Info	Fixed
Unused modifier	zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol	Info	Fixed
Calldata location can be used instead of memory	zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol zkbob-contracts/src/zkbob/ZkBobPool.sol zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol bob-cdp-contracts/src/Vault.sol	Info	Fixed
Variable can be set as immutable	bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol	Info	Acknowledged
Variable can be cached	zkbob-contracts/src/BobVault.sol zkbob-	Info	Fixed

Issue	Contract	Risk Level	Status
	contracts/src/zkbob/ZkBobDirectDepositQueue.sol		
Missing check that batch_deposit_verifier is contract	zkbob-contracts/src/zkbob/ZkBobPool.sol	Info	Fixed
Missing isContract check for EIP1967 proxy implementation	zkbob-contracts/src/proxy/EIP1967Proxy.sol	Info	Acknowledged
Missing 0x0 check on input addresses	zkbob-contracts/src/utils/Claimable.sol zkbob-contracts/src/utils/UniswapV3Seller.sol bob-cdp-contracts/src/Vault.sol	Info	Fixed
Unnecessary calculations in BaseERC20	zkbob-contracts/src/token/BaseERC20.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Mint/burn quota bypass in BalancedMinter

Risk Level: High

Status: Fixed in the commit c62404f4. Code corrected, values of 2**127 or larger are no longer allowed.

Contracts:

- zkbob-contracts/src/minters/BalancedMinter.sol

Location: Function: `_beforeMint, _beforeBurn`.

Description:

`BalanceMinter` contract limits the amount of minted tokens which must not exceed `mintQuota` (`burnQuota` for burned tokens respectively). Because of unsafe type casting this quota can be bypassed when `_amount` argument value is more than 2 ** 127.

```
function _beforeMint(uint256 _amount) internal override {
    int128 amount = int128(uint128(_amount));
    unchecked {
        require(mintQuota >= amount, "BalancedMinter: exceeds minting quota");
        (mintQuota, burnQuota) = (mintQuota - amount, burnQuota + amount);
    }
}
```

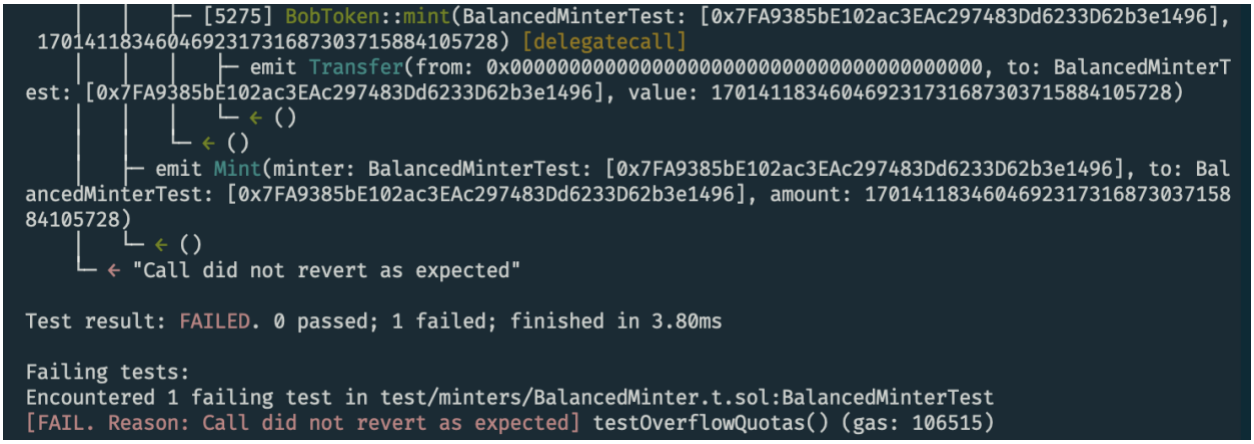
Remediation:

Ensure that `amount` after casting to `int128` is greater than zero.

Proof of concept:

```
function testOverflowQuotas() public {
    bob.mint(address(this), 200 ether);
    minter.setMinter(address(this), true);

    vm.expectRevert("BalancedMinter: exceeds minting quota");
    minter.mint(address(this), 2 ** 127);
}
```



`forge test --match-contract BalancedMinterTest --match-test testOverflowQuotas -vvv`

5.2. Any token can be transferred from liquidation bot

Risk Level: High

Status: Fixed in the commit e8a507fa. Code corrected, recommended check was added.

Contracts:

- cdp-nft-liqbot/contracts/src/Bot.sol

Location: Lines: 71. Function: `onFlashLoan`.

Description:

zkBob CDP liquidation bot implements `IERC3156FlashBorrower` interface to get flash loans of `BOB` tokens from zkBob's `FlashMinter` contract in order to liquidate underwater CDP positions. However, it does not properly authorize caller in `onFlashloan` callback making it possible to initiate the flash loan from the `FlashMinter` instead of liquidation bot itself (function `liquidate`).

In an attack scenario a malicious flashloan is requested from `FlashMinter` such that `decoded.cdp` points to attacker's address, `decoded.debt` is any amount of `BOB` and `decoded.nfts` is an empty list. In a situation when the liquidation bot holds `BOB` tokens, the attacker may force it into approving an arbitrary amount of `BOB` tokens so that they can be transferred out from the liquidation bot.

Moreover, by crafting `decoded.swapAddresses` and `decoded.swapData` any external call can be performed, including a call to privileged function `approve` which can be used by the attacker to approve any ERC20 or ERC721 (e.g. Uniswap V3 positions) tokens that liquidation bot possesses.


```
/// @inheritdoc IERC3156FlashBorrower
ftrace | funcSig
function onFlashLoan(
    address initiator,
    address token,
    uint256 amount,
    uint256 fee,
    bytes calldata data
) external isAuthorized returns (bytes32) {
    FlashCallbackData memory decoded = abi.decode(data, (FlashCallbackData));

    // liquidate
    ICDP cdp = decoded.cdp;
    IERC20(token).approve(address(cdp), decoded.debt);
    cdp.liquidate(decoded.vaultId);

    // close all contracts positions
    for (uint256 i = 0; i < decoded.nfts.length; i++) {
        closeUniV3Position(decoded.nfts[i], decoded.positionManager);
    }

    // swapping all tokens for the bob
    executeCalls(decoded.swapAddresses, decoded.swapData);

    IERC20(token).approve(msg.sender, amount + fee);
    return keccak256("ERC3156FlashBorrower.onFlashLoan");
}
```

Remediation:

Add `initiator` check as per https://eips.ethereum.org/EIPS/eip-3156.

```
require(
    initiator == address(this),
    "FlashBorrower: Untrusted loan initiator"
);
```

This will not allow a flash loan to be requested outside of the liquidation bot.

5.3. Centralization risk

Risk Level: Medium

Status: Risk acknowledged, proxy upgradeability or similar features will be put behind a sufficient timelock in the future.

Description:

The ZkBobPool contract has allowances from users and may contain big amount of funds (up to 10 millions BOB on https://docs.zkbob.com/zkbob-overview/deposit-and-withdrawal-limits). Because ZkBobPool implementation is deployed beyond the proxy and there are no limitations on the owner's actions due to the owner of the proxy is not a timelock contract but rather a gnosis safe.

Remediation:

Consider creating a separate role for the critical actions to provide better transparency and decentralization.

5.4. Pool limitations lead to DoS

Risk Level: Medium

Status: Risks of limits management is well-understood and will be mitigated as far as possible by increasing enforced limits over time, separating user addresses across multiple limit tiers, implementing sufficient DoS protections in the sequencer codebase.

Description:

There are several caps on the pool that disable the main functions of the pool:

- `dailyDepositCap` - max cap on the daily deposits sum
- `dailyWithdrawalCap` - max cap on the daily withdrawal sum

If `dailyDepositCap` exceeds the cap, then nobody else will have the ability to deposit funds in the pool. The `dailyWithdrawalCap` works the same — if it exceeds the cap then no one can withdraw from the pool. These limitations could be used by a malicious actor to cause DoS on the main functions of the pool — to deposit and to withdraw.

The documentation says that a single deposit cannot exceed 10,000 BOB and all deposits from all addresses cannot exceed 300,000 BOB during a single 24-hour period. This means that if 30 addresses send 10,000 BOB tokens to the pool, then pool will not be able to receive any tokens for the rest of the day.

Also, the documentation says that all withdrawals from all addresses cannot exceed 100,000 BOB during a single 24 hour period. It means that if 10 address start to withdraw their previous deposits of 10,000 BOB, then pool will not be able to withdraw any tokens for the rest of the day.

This information is relevant for tier 0 (default tier). It means that anyone could cause DoS on withdrawals with 100,000 BOB using 10 addresses and cause DoS on deposits with 300,000 BOB using 30 addresses.

Remediation:

To remediate the issue of caps causing DoS on the main functions of the pool, some measures can be taken such as:

- Regularly monitoring the deposits and withdrawals to identify any suspicious activity
- Relayers may prioritize withdrawals for the addresses that passed KYC
- Developing an algorithm that automatically adjusts the daily caps based on the previous day's activity to prevent anyone from abusing the system. Implementing these measures will help in mitigating the risk of DoS attacks caused by caps on the pool.

5.5. Direct deposit limits do not match the pool limits

Risk Level: Medium

Status: Code describes an intended system behavior. Limits abuse can be mitigated by setting direct deposit limits for default tier to 0 and by implementing sufficient DoS protections in the sequencer codebase.

Contracts:

- zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol

Description:

There are two methods to deposit funds into `ZkBobPool`:

1) use the `transact()` function
2) use the new direct deposit feature via `appendDirectDeposits()`.

The first method has a lot of restrictions that help to protect against large deposits which may be sourced from illegal activity, the limits are stored in `structs` called `TierLimits` and `TierStats`.

Below are the values of `TierLimits`:

- `tvlCap` - max cap on the entire pool TVL
- `dailyDepositCap` - max cap on the daily deposits sum
- `dailyUserDepositCap` - max cap on the daily deposits sum for single user
- `depositCap` - max cap on a single deposit

Below are the values of `TierStats`:

- `dailyDeposit` - sum of all deposits during given day
- `dailyWithdrawal` - sum of all withdrawals during given day

The second type of deposits (direct deposits) have only two types of restrictions in `TierLimits`:

- `directDepositCap` - max cap on a single direct deposit
- `dailyUserDirectDepositCap` - max cap on the daily direct deposits sum for single user

Thus, it can be seen that direct deposits miss a lot of restrictions and allow to bypass deposit limitations for the pools. It's even possible to exceed the max cap on the entire pool TVL, because it is missing this check from `_processTVLChange()` function:

```
require(uint256(s1.tvl) <= uint256(tl.tvlCap) * PRECISION, "ZkBobAccounting:
tvl cap exceeded");
```

Remediation:

Consider implementing restrictions for direct deposits as it was done to regular deposits.

5.6. Direct deposits DoS

Risk Level: Medium

Status: Since the operator manager in these contracts is only required for fast-track refunds, added a warning comment to avoid usage of permissionless operator manager contracts. See the commit c62404f4.

Contracts:

- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol,
- zkbob-contracts/src/zkbob/manager/MutableOperatorManager.sol
- zkbob-contracts/src/zkbob/manager/SimpleOperatorManager.sol

Location: Function: `refundDirectDeposit`.

Description:

There is a `refundDirectDeposit()` function in ZkBobDirectDepositQueue that can be used to refund funds sent via the direct deposits function of the contract. The refund will pass successfully if it was called by an operator or due to a passed timeout.

However, the operator check could be bypassed if the `operator` variable in `operatorManager` is set to zero - `address(0)` allows anyone to be an operator. This will open an opportunity to DoS - every direct deposit could be refunded before the relayer starts to process it.

Remediation:

Consider removing the functionality of allowing anybody to be the operator in `ZkBobDirectDepositQueue`.

5.7. User controlled calls in collectAndIncreaseAmount

Risk Level: Medium

Status: Fixed in the commit 5cc1c336. Code corrected, added the suggested check.

Contracts:

- bob-cdp-contracts/src/Vault.sol

Location: Lines: 578. Function: `collectAndIncreaseAmount`.

Description:

The public function `collectAndIncreaseAmount` accepts two arguments: `collectParams` and `increaseLiquidityParams` which represent parameters to collect fees from a Uniswap v3 pool and to increase pool liquidity respectively. The function extracts `vaultId` using `tokenId` (Uniswap v3 position NFT) from the `collectParams` and validates that the user is authorized to perform fee collection if he or she is the owner of the vault.

```
uint256 tokenId = collectParams.tokenId;
uint256 vaultId = vaultIdByNft[tokenId];
_requireVaultAuth(vaultId);
```

However, the `tokenId` from `increaseLiquidityParams` is not validated. It is possible to submit a different `tokenId` such that `getPositionTokens` will return arbitrary token addresses that are not

whitelisted by the zkBob CDP protocol. To do so an attacker can deploy two ERC20 tokens, create a Uniswap v3 pool with these tokens and open a position in this pool. After that he or she may submit `increaseLiquidityParams` with this position's `tokenId` which will make `Vault` contract to call these token addresses:

```
599: IERC20(token0).transferFrom(msg.sender, address(this),
increaseLiquidityParams.amount0Desired);
600: IERC20(token1).transferFrom(msg.sender, address(this),
increaseLiquidityParams.amount1Desired);
610: IERC20(token0).transfer(msg.sender,
increaseLiquidityParams.amount0Desired - depositedAmount0);
614: IERC20(token1).transfer(msg.sender,
increaseLiquidityParams.amount1Desired - depositedAmount1);
```

Right now, the impact is as follows:

3) `Vault` contract can be tricked into approving any amount of any token to `UniV3PositionManager` contract via `_checkAllowance`.
4) user controllable calls can re-enter the `Vault` contract which does not have reentrancy guards

Remediation:

Check that `tokenId` from `collectParams` is equal to `tokenId` from `increaseLiquidityParams`:

```
require(collectParams.tokenId == increaseLiquidityParams.tokenId);
```

5.8. Missing return value check of the ERC20 transfers

Risk Level: Medium

Status: Fixed in the commit c62404f4. Code corrected, replaced with usage of SafeERC20 library.

Contracts:

- zkbob-contracts/src/utils/UniswapV3Seller.sol

Description:

The missing return value check of the ERC20 transfer can result in loss of funds, because some ERC20 tokens indicate a failed transfer by the return value instead of a `revert`. The code snippets in the description show instances where the return value of the `transfer` function is not checked.

```
src/utils/UniswapV3Seller.sol:
  70:           IERC20(token0).transfer(msg.sender, refund);
```

Remediation:

To fix this issue, it is recommended to always check the return value of the `transfer`/`transferFrom` functions and handle the failure case appropriately, such as by reverting the transaction.

References:

- https://github.com/d-xo/weird-erc20#no-revert-on-failure
- https://github.com/crytic/building-secure-contracts/blob/master/development-guidelines/token_integration.md

5.9. Wrong parameter passed to event

Risk Level: Low

Status: Fixed in the commit c62404f4. Code corrected.

Contracts:

- zkbob-contracts/src/minters/BalancedMinter.sol

Location: Lines: 32. Function: `adjustQuotas`.

Description:

There is an `adjustQuotas()` function in the BalancedMinter contract that can be used to adjust mint/burn quotas for the given address. It stores two calculated values in the memory `newMintQuota` and `newBurnQuota` write them to storage and emits a `UpdateQuotas(int128 mintQuota, int128 burnQuota)` event. However, `newBurnQuota` was passed to the `UpdateQuotas` event as the first `mintQuota` argument rather than `newMintQuota`.

```
zkbob-contracts/src/minters/BalancedMinter.sol:
  32:         emit UpdateQuotas(newBurnQuota, newBurnQuota);
```

Remediation:

Consider changing the first argument from `newBurnQuota` to `newMintQuota`.

5.10. Missing workflow invariants

Risk Level: Low

Status: Fixed in the commit 5cc1c336. Code corrected, added usage of ReentrancyGuard.

Description:

In the `Vault` contract of the zkBob's CDP implementation there are multiple public state changing functions that are responsible for a particular action:

- `openVault`
- `closeVault`
- `burnVault`
- `depositCollateral`
- `withdrawCollateral`
- `mintDebt`
- `burnDebt`
- `liquidate`
- `withdrawOwed`
- `decreaseLiquidity`
- `collect`
- `collectAndIncreaseAmount`

Although the correct vault state is ensured by checking the vault health parameters, the state transitions are not enforced declaratively. For instance, it does not make sense to `openVault` and `closeVault` in one transaction or `depositCollateral` and `withdrawCollateral` at the same time. It is a good practice to allow a single meaningful action per transaction to avoid possible external oracle exploits targeted at price manipulation.

Remediation:

Consider restricting sequences of allowed calls in the same transaction as well as using a `ReentrancyGuard` from https://github.com/OpenZeppelin/openzeppelin-contracts.

References:

- https://twitter.com/MixBytes/status/1634252823868379138

5.11. Infinite allowance in _checkAllowance

Risk Level: Low

Status: Fixed in the commit 5cc1c336. Code corrected, added usage of exact value approvals.

Contracts:

- bob-cdp-contracts/src/Vault.sol

Location: Lines: 870. Function: _checkAllowance.

Description:

The collectAndIncreaseAmount function allows to collect fees and increase amount of liquidity in a Uniswap v3 pool simultaneously. In order to increase liquidity, the Vault has to give allowance to UniV3PositionManager to spend tokens from the Vault. This is done via _checkAllowance function:

```
function _checkAllowance(
    address targetToken,
    uint256 amount,
    address target
) internal {
    if (IERC20(targetToken).allowance(address(this), target) < amount) {
        IERC20(targetToken).approve(target, type(uint256).max);
    }
}
```

The function gives maximum allowance regardless of the required amount. It is a good practice to only allow only necessary amount even considering the fact that UniV3PositionManager is a trusted contract.

Remediation:

Consider allowing only necessary amount:

```
if (IERC20(targetToken).allowance(address(this), target) < amount) {
    IERC20(targetToken).approve(target, amount);
}
```

5.12. No events emitted on state changes

Risk Level: Low

Status: Fixed in the commits c62404f4 and 5cc1c336. Code corrected, added missing events.

Contracts:

- bob-cdp-contracts/src/oracles/UniV3Oracle.sol
- zkbob-contracts/src/minters/BaseMinter.sol
- zkbob-contracts/src/minters/FlashMinter.sol
- zkbob-contracts/src/minters/SurplusMinter.sol

Description:

Admin activities should be logged. This makes it easier to validate that admin is not abusing privileges before interacting with the application.

bob-cdp-contracts/src/oracles/UniV3Oracle.sol:

- `setMaxPriceRatioDeviation()`

zkbob-contracts/src/minters/BaseMinter.sol:

- `setMinter()`

zkbob-contracts/src/minters/FlashMinter.sol

- `updateConfig()`

zkbob-contracts/src/minters/SurplusMinter.sol

- `setMinter()`

Remediation:

Consider emitting events for important state changes.

5.13. Differences in Oracle implementations

Risk Level: Low

Status: Fixed in the commits 5cc1c336 and bf5a4161. Code corrected, unified the oracle implementations.

Contracts:

- bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol
- bob-cdp-contracts/src/oracles/UniV3Oracle.sol

Description:

Contracts `QuickswapV3Oracle` and `UniV3Oracle` are implementing the same logic but have different functionality. In `UniV3Oracle`, the admin can change the maximum price deviation through the `setMaxPriceRatioDeviation` function, while in `QuickswapV3Oracle` this option is not available.

Remediation:

Consider updating the `QuickswapV3Oracle` contract to include the `setMaxPriceRatioDeviation` function for consistency with the `UniV3Oracle` contract. Alternatively, document the differences in functionality between the two contracts for future developers.

5.14. Non-optimal increment/decrement

Risk Level: Info

Status: Fixed in the commits c62404f4 and bf5a4161. Code corrected.

Contracts:

- zkbob-contracts/src/token/ERC20Recovery.sol
- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol
- zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol
- bob-cdp-contracts/src/Vault.sol
- bob-cdp-contracts/src/oracles/ChainlinkOracle.sol

Description:

The difference between the prefix increment and postfix increment expression lies in the return value of the expression.

The prefix increment expression (`++i`) returns the *updated* value after it's incremented. The postfix increment expression (`i++`) returns the *original* value.

The prefix increment expression is cheaper in terms of gas.

Consider using the prefix increment expression whenever the return value is not needed.

There are the following occurrences:

```
zkbob-contracts/src/token/ERC20Recovery.sol:
  130:         for (uint256 i = 0; i < _values.length; i++) {
  193:         for (uint256 i = 0; i < _accounts.length; i++) {
```

```
zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol:
  128:         for (uint256 i = 0; i < count; i++) {
  216:         for (uint256 i = 0; i < _indices.length; i++) {

zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol:
  190:         s0.txCount++;
  351:         for (uint256 i = 0; i < _users.length; i++) {

bob-cdp-contracts/src/Vault.sol:
  388:         for (uint256 i = 0; i < vaultNfts.length; i++) {
  722:         for (uint256 i = 0; i < depositors.length; i++) {
  730:         for (uint256 i = 0; i < depositors.length; i++) {
  738:         for (uint256 i = 0; i < liquidators.length; i++) {
  746:         for (uint256 i = 0; i < liquidators.length; i++) {

bob-cdp-contracts/src/oracles/ChainlinkOracle.sol:
  171:         for (uint256 i = 0; i < tokens.length; i++) {
```

Remediation:

An example of a not optimized code:

```
for (uint i = 1; i <= len; i++) {
```

Consider the following example to save gas:

```
for (uint i = 1; i <= len; ++i) {
```

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g012—use-prefix-increment-instead-of-postfix-increment-if-possible

5.15. Unnecessary checked arithmetic in loop

Risk Level: Info

Status: Potential optimizations acknowledged, no actions would be taken to reduce code bloat.

Contracts:

- zkbob-contracts/src/token/ERC20Recovery.sol
- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol
- zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol

- bob-cdp-contracts/src/Vault.sol
- bob-cdp-contracts/src/oracles/ChainlinkOracle.sol
- bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol
- bob-cdp-contracts/src/oracles/UniV3Oracle.sol
- bob-cdp-contracts/src/token/ERC721/ERC721.sol

Description:

Use unchecked blocks where overflow/underflow is impossible. There are several cycles where arithmetic checks are not necessary:

```
zkbob-contracts/src/token/ERC20Recovery.sol:
  130:         for (uint256 i = 0; i < _values.length; i++) {
  193:         for (uint256 i = 0; i < _accounts.length; i++) {

zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol:
  128:         for (uint256 i = 0; i < count; i++) {
  216:         for (uint256 i = 0; i < _indices.length; i++) {

zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol:
  351:         for (uint256 i = 0; i < _users.length; i++) {

bob-cdp-contracts/src/Vault.sol:
  388:         for (uint256 i = 0; i < vaultNfts.length; i++) {
  722:         for (uint256 i = 0; i < depositors.length; i++) {
  730:         for (uint256 i = 0; i < depositors.length; i++) {
  738:         for (uint256 i = 0; i < liquidators.length; i++) {
  746:         for (uint256 i = 0; i < liquidators.length; i++) {
  820:         for (uint256 i = 0; i < vaultNfts.length; ++i) {
  931:         for (uint256 i = 0; i < vaultNfts.length; ++i) {

bob-cdp-contracts/src/oracles/ChainlinkOracle.sol:
  171:         for (uint256 i = 0; i < tokens.length; i++) {

bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol:
  121:         for (uint256 i = 0; i < 2; ++i) {

bob-cdp-contracts/src/oracles/UniV3Oracle.sol:
  117:         for (uint256 i = 0; i < 2; ++i) {

bob-cdp-contracts/src/token/ERC721/ERC721.sol:
  465:         for (uint256 i = 0; i < 32; ++i) {
```

Remediation:

An example of a not optimized code:

```
for (uint i; i < len; ++i) {
    // ...
}
```

Consider the following example to save gas:

```
for (uint i; i < len; ) {
    // ...
    unchecked { ++i; }
}
```

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g011—unnecessary-checked-arithmetic-in-for-loop

5.16. Non-optimal conditional statements

Risk Level: Info

Status: Potential optimizations acknowledged, no actions would be taken to reduce code bloat.

Contracts:

- zkbob-contracts/src/libraries/ZkAddress.sol
- zkbob-contracts/src/utils/EIP712.sol
- zkbob-contracts/src/zkbob/ZkBobPool.sol
- zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol
- bob-cdp-contracts/src/Vault.sol
- bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol
- bob-cdp-contracts/src/utils/VaultAccessControl.sol

Description:

Using nested is cheaper than using && multiple check combinations. There are more advantages, such as easier to read code and better coverage reports.

```
zkbob-contracts/src/libraries/ZkAddress.sol:
  74:         } else if (len < 64 && len > 46) {

zkbob-contracts/src/utils/EIP712.sol:
  69:         if (address(this) == _CACHED_THIS && block.chainid ==
_CACHED_CHAIN_ID) {
```

```
zkbob-contracts/src/zkbob/ZkBobPool.sol:
  225:             require(transfer_token_delta > 0 && energy_amount == 0,
"ZkBobPool: incorrect deposit amounts");
  229:             require(token_amount == 0 && energy_amount == 0,
"ZkBobPool: incorrect transfer amounts");
  232:             require(token_amount <= 0 && energy_amount <= 0,
"ZkBobPool: incorrect withdraw amounts");
  252:             require(transfer_token_delta > 0 && energy_amount == 0,
"ZkBobPool: incorrect deposit amounts");

zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol:
  168:         if (s0.txCount > 0 && curSlot - s0.tailSlot > WEEK_SLOTS) {
  338:         if (_configuredTier == 0 && address(kycProvidersMgr) !=
address(0)) {
  340:             if (kycPassed && tiers[tier].limits.tvlCap > 0) {

bob-cdp-contracts/src/Vault.sol:
  336:         if (!isPublic && !_depositorsAllowlist.contains(msg.sender)) {
  466:         if (!isLiquidatingPublic &&
!_liquidatorsAllowlist.contains(msg.sender)) {
  832:                 if (limitType == SAFE_BORROW_LIMIT && !deviationSafety)
{
  889:         if (!isPublic && !_depositorsAllowlist.contains(caller)) {
  899:         if (vaultNfts.length == 0 && vaultRegistry.minterOf(vaultId) !=
address(this)) {

bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol:
  98:                 if (1 ether - maxPriceRatioDeviation < deviation &&
deviation < 1 ether + maxPriceRatioDeviation) {

bob-cdp-contracts/src/utils/VaultAccessControl.sol:
  42:         if (!isAdmin(msg.sender) && !isOperator(msg.sender)) {
```

Remediation:

Use split **if** statements instead of **&&.**

5.17. Unused function parameter

Risk Level: Info

Status: Acknowledged, no gas usage improvements were noted, hence no code change.

Contracts:

- zkbob-contracts/src/minters/BaseMinter.sol
- zkbob-contracts/src/minters/SurplusMinter.sol

Description:

There are several functions where some of the variables are unused:

- `_data` in BaseMinter.sol
- `_from` in SurplusMinter.sol
- `operator` in Vault.sol

```
zkbob-contracts/src/minters/BaseMinter.sol:
  89:     function onTokenTransfer(address _from, uint256 _amount, bytes
calldata _data) external override returns (bool) {

zkbob-contracts/src/minters/SurplusMinter.sol:
  61:     function onTokenTransfer(address _from, uint256 _amount, bytes
calldata _data) external override returns (bool) {

bob-cdp-contracts/src/Vault.sol:
  536:     function onERC721Received(
  537:         address operator,
  538:         address from,
  539:         uint256 tokenId,
  540:         bytes memory data
  541:     ) external onlyUnpaused returns (bytes4) {
```

Remediation:

Consider commenting out the variable name:

```
function onTokenTransfer(address _from, uint256 _amount, bytes calldata /*
_data */) {
    // ...
}
```

5.18. Cache array length outside the loop

Risk Level: Info

Status: Fixed in the commits 5cc1c336 and bf5a4161. Code corrected, array length is cached where it leads to an SLOAD.

Contracts:

- /bob-cdp-contracts/src/Vault.sol
- /bob-cdp-contracts/src/Vault.sol
- /bob-cdp-contracts/src/oracles/ChainlinkOracle.sol

Description:

Caching the array length outside a loop saves reading it on each iteration, as long as the array's length is not changed during the loop.

There are the following instances:

- /bob-cdp-contracts/src/Vault.sol:820
- /bob-cdp-contracts/src/Vault.sol:931
- /bob-cdp-contracts/src/oracles/ChainlinkOracle.sol:171

Remediation:

Example of a not optimized code:

```
for (uint256 i = 0; i < tokens.length; i++) {
```

Consider saving array length before the loop:

```
uint256 l = tokens.length;
for (uint256 i = 0; i < l; i++) {
```

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g002—cache-array-length-outside-of-loop
- https://github.com/code-423n4/2021-11-badgerzaps-findings/issues/36

5.19. String error messages used instead of custom errors

Risk Level: Info

Status: Acknowledged, but no code changes will be made. Suggestions will be taken into account for contracts developed in the future.

Contracts:

- bob-cdp-contracts/src/token/ERC721/ERC721.sol
- bob-cdp-contracts/src/token/ERC721/ERC721Enumerable.sol
- zkbob-contracts/src/BobVault.sol
- zkbob-contracts/src/minters/BalancedMinter.sol
- zkbob-contracts/src/minters/BaseMinter.sol

- zkbob-contracts/src/minters/DebtMinter.sol
- zkbob-contracts/src/minters/FaucetMinter.sol
- zkbob-contracts/src/minters/FlashMinter.sol
- zkbob-contracts/src/minters/SurplusMinter.sol
- zkbob-contracts/src/proxy/EIP1967Admin.sol
- zkbob-contracts/src/proxy/EIP1967Proxy.sol
- zkbob-contracts/src/token/BaseERC20.sol
- zkbob-contracts/src/token/ERC20Blocklist.sol
- zkbob-contracts/src/token/ERC20MintBurn.sol
- zkbob-contracts/src/token/ERC20Permit.sol
- zkbob-contracts/src/token/ERC20Recovery.sol
- zkbob-contracts/src/token/ERC677.sol
- zkbob-contracts/src/utils/Claimable.sol
- zkbob-contracts/src/utils/Ownable.sol
- zkbob-contracts/src/utils/UniswapV3Seller.sol
- zkbob-contracts/src/yield/AAVEYieldImplementation.sol
- zkbob-contracts/src/yield/YieldConnector.sol
- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol
- zkbob-contracts/src/zkbob/ZkBobPool.sol
- zkbob-contracts/src/zkbob/ZkBobPoolETH.sol
- zkbob-contracts/src/zkbob/manager/MutableOperatorManager.sol
- zkbob-contracts/src/zkbob/manager/SimpleOperatorManager.sol
- zkbob-contracts/src/zkbob/manager/kyc/SimpleKYCProviderManager.sol
- zkbob-contracts/src/zkbob/utils/KycProvidersManagerStorage.sol
- zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol

Description:

The contracts make use of the `require()` to emit an error. While this is a perfectly valid way to handle errors in Solidity, it is not always the most efficient.

Remediation:

Consider using custom errors as they are more gas efficient while allowing developers to describe the error in detail using NatSpec.

References:

- https://blog.soliditylang.org/2021/04/21/custom-errors/

5.20. Inconsistent variable naming style

Risk Level: Info

Status: Fixed in the commit 5cc1c336. Code corrected, suggested naming changes applied.

Contracts:

- bob-cdp-contracts/src/interfaces/ICDP.sol
- bob-cdp-contracts/src/Vault.sol

Description:

Some variables are stored in the fixed point representation and have different naming styles. It causes confusion and can lead to bugs in the future.

In `Vault.sol` :

- `stabilisationFeeRateD` is a `stabilisationFeeRate * DEBT_DENOMINATOR`

In`ICDP.sol` protocol params:

- `liquidationPremiumDis a liquidationPremium* DENOMINATOR liquidationFeeDis aliquidationFee* DENOMINATOR`

In `ICDP.sol` pool params:

- `liquidationThresholdis` an actual `liquidationThreshold* DENOMINATOR`
- `borrowThresholdis` an actual `borrowThreshold* DENOMINATOR`

Remediation:

Consider using D18 and D9 suffixes for such variables.

5.21. Outdated OpenZeppelin library

Risk Level: Info

Status: Fixed in the commits c62404f4 and 5cc1c336. Library updated to the up-to-date `v4.8.3` tag.

Contracts:

- zkbob-contracts/lib/@openzeppelin

Description:

There is an outdated OpenZeppelin library in dependencies (version 4.7.0). This library contains several known vulnerabilities that may have a security impact when new features are added to zkBob.

Remediation:

Consider updating the dependencies on time to avoid the risk of known vulnerabilities.

5.22. Missing zero transfer token amount check in ZkBobPool

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected, the check moved outside of the function, but kept inside the `_recordOperation` to preserve remaining storage writes.

Contracts:

- zkbob-contracts/src/zkbob/ZkBobPool.sol

Description:

There is a check in the `_processTVLChange()` function from the `ZkBobAccounting` contract that validates and stores the value that was deposited/withdrawn from the pool. In case this amount is equal to zero this function does nothing.

```
zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol:
  201:         if (_txAmount == 0) {
  202:             return;
  203:         }
```

To save gas and avoid calls with `_transfer_token_amount()` equal to zero, this check might be moved to the `ZkBobPool` contract.

Remediation:

Consider removing this check from `ZkBobAccounting` and adding it before the `_recordOperation()` call.

```
zkbob-contracts/src/zkbob/ZkBobPool.sol:
  193           int256 transfer_token_delta = _transfer_token_amount();
  194:          require(transfer_token_delta != 0, "ZkBobPool: zero transfer
amount");
  195           (,, uint256 txCount) = _recordOperation(user,
transfer_token_delta);
```

5.23. Unused variables

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected, unused constant removed.

Contracts:

- bob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol

Location: Lines: 24.

Description:

Constant `MAX_POOL_ID` in `ZkBobDirectDepositQueue` is never used.

Remediation:

Consider removing the unused constant.

5.24. Local variable shadowing

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected, variable renamed.

Contracts:

- zkbob-contracts/src/minters/FlashMinter.sol

Location: Lines: 107. Function: `flashLoan`.

Description:

The local variable `fee` in the function `flashLoan` of the `FlashMinter` contract shadows the state variable with the same name defined on the 24th line.

Remediation:

Consider renaming the local variable `fee` in the `flashLoan` function.

5.25. Unused modifier

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected, unused modifier removed.

Contracts:

- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol

Location: Lines: 64. Function: `onlyOperator`.

Description:

The modifier `onlyOperator` in `ZkBobDirectDepositQueue` is never used.

Remediation:

Consider removing unused modifier.

5.26. Calldata location can be used instead of memory

Risk Level: Info

Status: Partially fixed in the commit bf5a4161. Code partially corrected, the `calldata` modifier is now used where appeared to be possible.

Contracts:

- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol
- zkbob-contracts/src/zkbob/ZkBobPool.sol
- zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol
- bob-cdp-contracts/src/Vault.sol

Location: Function: `_recordDirectDeposit` `setUsersTier` `_setUsersTier` `onERC721Received`.

Description:

In the function, `_recordDirectDeposit` argument `_rawZkAddress` is defined as `bytes memory` while `calldata` location should be used because `_rawZkAddress` is not modified.

```
zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol:
  239:     function _recordDirectDeposit(
  240:         address _sender,
```

```
  241:         address _fallbackReceiver,
  242:         uint256 _amount,
  243:         bytes memory _rawZkAddress
  244:     )
```

In the `setUsersTier()` and `_setUsersTier()` functions argument `_users` is defined as `address[] memory` while `calldata` location should be used because `_users` is not modified.

```
zkbob-contracts/src/zkbob/ZkBobPool.sol:
  393:     function setUsersTier(uint8 _tier, address[] memory _users)
external onlyOwner {
zkbob-contracts/src/zkbob/utils/ZkBobAccounting.sol:
  347:     function _setUsersTier(uint8 _tier, address[] memory _users)
internal {
```

In the function, `onERC721Received` argument `data` is defined as `bytes memory` while `calldata` location should be used because `data` is not modified.

```
bob-cdp-contracts/src/Vault.sol:
  536:     function onERC721Received(
  537:         address operator,
  538:         address from,
  539:         uint256 tokenId,
  540:         bytes memory data
  541:     ) external onlyUnpaused returns (bytes4) {
```

Remediation:

Change `bytes memory` to `bytes calldata` to optimize gas usage. This change is recommended when a function is not modifying the value of an argument and only reads it. Using `calldata` instead of `memory` can improve gas efficiency.

5.27. Variable can be set as immutable

Risk Level: Info

Status: No longer valid, after changes applied in “Differences in Oracle implementations”.

Contracts:

- bob-cdp-contracts/src/oracles/QuickswapV3Oracle.sol

Location: Lines: 35.

Description:

State variable `maxPriceRatioDeviation` from `QuickswapV3Oracle` contract cannot be changed but is not set as `immutable`.

Remediation:

Since `maxPriceRatioDeviation` is not expected to change during the execution of the contract, it is a good practice to define it as an immutable variable. This can be achieved by adding the `immutable` keyword before the variable declaration. By doing so, the variable will be set at compile-time instead of run-time, reducing gas costs and improving performance.

5.28. Variable can be cached

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected.

Contracts:

- zkbob-contracts/src/BobVault.sol
- zkbob-contracts/src/zkbob/ZkBobDirectDepositQueue.sol

Location: Lines: 572. Function: `farm`.

Description:

`token.yield` storage variable is used several times in the `farm()` function execution BobVault.sol#L572

`directDepositTimeout` storage variable may be used several times in the loop during `refundDirectDeposit()` function execution ZkBobDirectDepositQueue#221

Remediation:

Consider caching `token.yield` and `directDepositTimeout` variables to reduce the number of times the storage variable is accessed. This can improve gas optimization and potentially reduce transaction costs. One way to cache it is to create a local variable and set it equal to the storage variable at the beginning of the function, then use the local variable throughout the rest of the function instead of accessing the storage variable directly.

5.29. Missing check that batch_deposit_verifier is contract

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected.

Contracts:

- zkbob-contracts/src/zkbob/ZkBobPool.sol

Location: Lines: 56. Function: `constructor`.

Description:

`ZkBobPool` contract is missing check in the constructor that `_batch_deposit_verifier` variable is a contract.

Remediation:

Consider adding check for that variable as is done for other addresses.

```
require(Address.isContract(_batch_deposit_verifier), "ZkBobPool: not a
contract");
```

5.30. Missing isContract check for EIP1967 proxy implementation

Risk Level: Info

Status: No changes, as multiple proxy contracts were already deployed and actively used.

Contracts:

- zkbob-contracts/src/proxy/EIP1967Proxy.sol

Location: Lines: 125. Function: `_setImplementation`.

Description:

The missing `isContract` check in `_setImplementation` leaves the contract open to potential errors when setting a new implementation. It is recommended to add a check that verifies that the target address is a contract before allowing the proxy admin to set a new implementation.

Remediation:

To remediate this issue, a simple if statement can be added to the contract's code that checks if the target address is a contract using the `isContract` function from the OpenZeppelin library. If the target is not a contract, the function should revert with an appropriate error message.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.3/contracts/proxy/ERC1967/ERC1967Upgrade.sol#L42

5.31. Missing 0x0 check on input addresses

Risk Level: Info

Status: Partially fixed in the commit 5cc1c336. Code partially corrected.

Contracts:

- zkbob-contracts/src/utils/Claimable.sol
- zkbob-contracts/src/utils/UniswapV3Seller.sol
- bob-cdp-contracts/src/Vault.sol

Description:

The are multiple occurrences when an input address argument is allowed to be a zero address which may lead to a loss of funds or lost access:

- zkbob-contracts/src/utils/Claimable.sol (argument _to)

```
  38:     function claimTokens(address _token, address _to) external virtual
onlyClaimingAdmin {
  39:         if (_token == address(0)) {
  40:             payable(_to).transfer(address(this).balance);
  41:         } else {
  42:             uint256 balance = IERC20(_token).balanceOf(address(this));
  43:             IERC20(_token).transfer(_to, balance);
  44:         }
  45:     }
```

- zkbob-contracts/src/utils/UniswapV3Seller.sol (arguments _swapRouter, _quoter, _token0, _token1)

```
  26:     constructor(address _swapRouter, address _quoter, address _token0,
uint24 _fee0, address _token1, uint24 _fee1) {
  27:         IERC20(_token0).approve(_swapRouter, type(uint256).max);
  28:         swapRouter = ISwapRouter(_swapRouter);
  29:         quoter = IQuoter(_quoter);
  30:         WETH = IWETH9(IPeripheryImmutableState(_swapRouter).WETH9());
  31:         token0 = _token0;
  32:         fee0 = _fee0;
```

```
  33:         token1 = _token1;
  34:         fee1 = _fee1;
  35:     }
```

- bob-cdp-contracts/src/Vault.sol (argument _minter)

```
constructor(
    INonfungiblePositionManager positionManager_,
    INFTOracle oracle_,
    address treasury_,
    address token_,
    address minter_,
    address vaultRegistry_
) {
    if (
        address(positionManager_) == address(0) ||
        address(oracle_) == address(0) ||
        address(treasury_) == address(0) ||
        address(token_) == address(0) ||
        address(vaultRegistry_) == address(0)
    ) {
        revert AddressZero();
    }
    ...
}
```

Remediation:

Consider checking that input address is not zero

5.32. Unnecessary calculations in BaseERC20

Risk Level: Info

Status: Fixed in the commit c62404f4. Code corrected.

Contracts:

- zkbob-contracts/src/token/BaseERC20.sol

Description:

The expression `1 << 255` is used many times and can be pre-calculated in the following occurrences:

```
zkbob-contracts/src/token/BaseERC20.sol:
  110:         require(balance < 1 << 255, "ERC20: account frozen");
  118:         require(balance < 1 << 255, "ERC20: account frozen");
```

```
133:         return _balances[_account] >= 1 << 255;
137:         _balances[_account] |= 1 << 255;
141:         _balances[_account] &= (1 << 255) - 1;
```

Remediation:

The expression `1 << 255` can be pre-calculated and stored in a constant variable.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.