HybridVault
Audit Report

Tue Jul 23 2024

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

HybridVault Audit Report

1 Executive Summary

1.1 Project Information

Description	A decentralized lending and collateral vault protocol
Type	DeFi
Auditors	ScaleBit
Timeline	Fri Jul 12 2024 - Tue Jul 23 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/KiloExPerp/HybridVault.git
Commits	56833b25d05a6182cd8f5f4eeaafda2348af3455 636593b140ce1ed87c9cde3b0b898f0cdfa7353e 1f12e45da443fab2c310acd9bea682cac076bf2e

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
PMA	contracts/libraries/PercentageMath.sol	5f9702d0e7e4aabbd7735028ba3b3e4ff0a5ed38
OOGU	contracts/access/OperatorOwnerGovernableUpgradeable.sol	f0edeac24f9b0dbbe54c3d98f2b0c8f00fc45da0
OGU	contracts/access/OwnerGovernableUpgradeable.sol	3d4e40d9e82b4b1d5fef28527577b95b85841e43
IOTPF	contracts/interfaces/IOpenTradesPnlFeed.sol	dd68fc4094ed0413972d446e4845cbe290dcd0b5
IKS	contracts/interfaces/IKiloStorage.sol	0f53f88300b2c8b5aef579565646b8ec31f1c26e
IPR	contracts/interfaces/IPendingReward.sol	aff851136859120eed5083d0c63c27593f0cc524
INF	contracts/interfaces/INft.sol	6e8930cd6492d01e221fdcd6f7103013a242ab9d
IKTLDND	contracts/interfaces/IKTokenLockedDepositNftDesign.sol	194c6156e8e15c61ed55bbf95f99344e1c9417a2
IOR	contracts/interfaces/IOracle.sol	ebdcce1a107c068c428d26e65b20eafd57a937e0
VUSD	contracts/hybridvault/VUSD.sol	9af44baeab1575b0849ad27369791d9bf1a3d170
IHT	contracts/hybridvault/IHToken.sol	a903b361e68232ca08520ee64cef8ae16aabe033

HTO	contracts/hybridvault/HToken.sol	df29b88b0fbc184ae56e739926b247ad18273c66
IVUSD	contracts/hybridvault/IVUSD.sol	ef4e0032913405e8bfc3206671932c18b2b7ba44
KERC4U	contracts/hybridvault/KiloERC4626Upgradeable.sol	692c2f3868db554f2209af6e8bce5acc4e357dc6
KTLDND	contracts/vaultv2/KTokenLockedDepositNftDesign.sol	0f1ae629eda97fb04bff0cf2d661ecfebfadbc3b
KTLDN	contracts/vaultv2/KTokenLockedDepositNft.sol	2b434c169073bc1b2ea95a7e3195bbc0ff0fc370
KTOPF	contracts/vaultv2/KTokenOpenPnlFeed.sol	4253abf692c29edbff8df446c5bad525e8aa770b
IKT	contracts/interfaces/IKToken.sol	22270cd58efff15cb866cdc16ff243cb56aa664f
VSR	contracts/core/VaultStakeReward.sol	9c3fc7fabb3b4c9aa12faf0ebdd8cd84c7cdf24b
HVL	contracts/hybridvault/HybridVaultLogic.sol	fe6170f9c4fb6c540f80cb830ae109240b0ac5eb
IPR1	contracts/hybridvault/IPriceRouter.sol	45c7737cac574cf32367f34ed7d3b6ffd7c29bf6
DTY	contracts/hybridvault/DataTypes.sol	7a12a46a10e9cb1c474f89373b6d151b05937214
IHV	contracts/hybridvault/IHybridVault.sol	9da0e461c25a48307e4d17e91c60dc945d473c3b
HVA	contracts/hybridvault/HybridVault.sol	e0ed559e5d9cb22a370f2e5bed6436fee7b5f40e

PRO	contracts/hybridvault/PriceRouter.sol	e2063c30aaf44c78a40c47e68ecf002b0afc23c7

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	6	2
Informational	1	0	1
Minor	3	2	1
Medium	3	3	0
Major	1	1	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by KiloEx to identify any potential issues and vulnerabilities in the source code of the HybridVault smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 8 issues of varying severity, listed below.

ID	Title	Severity	Status
HVA-1	The Condition Check in `setLiquidationBonus` Is Incorrect	Medium	Fixed
HVA-2	Only the Use of the Pyth oracle is Allowed to Incur a Fee	Minor	Fixed
PRO-1	Drain the Native Tokens from the PriceRouter	Major	Fixed
PRO-2	The Expired Price Check is Incorrect	Medium	Fixed
PRO-3	Users can Evade Updating the Oracle's Price	Medium	Fixed
PRO-4	Signature Replay	Minor	Acknowledged
PRO-5	Unused `isKeeper` Variable	Minor	Fixed
VSR-1	Redundant Inheritance	Informational	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the HybridVault Smart Contract :

Owner

- The Owner can configure hToken settings through the `configHToken` function.
- The Owner can set `liquidationBonus` through the `setLiquidationBonus` function.
- The Owner can set `quoteAssetsMinBp` through the `setQuoteAssetsMinBp` function.

Operator

- The Operator can set the LTV (Loan-to-Value) for HTokens through the `manageHTokenLtv` function.
- The Operator can perform rebalance through the `reBalance` function.
- The Operator can perform liquidation through the `liquidate` function.

User

- Users can make a deposit through the `deposit` function.
- Users can initiate a withdrawal request through the `makeWithdrawRequest` function.
- Users can cancel a withdrawal request through the `cancelWithdrawRequest` function.
- Users can redeem through the `redeem` function.
- Users can deposit assets with a discount and lock them for a specified duration through the `depositWithDiscountAndLock` function.
- Users can unlock a previously locked deposit and mint hTokens for the receiver through the `unlockDeposit` function.
- Users can distribute rewards through the `distributeReward` function.
- Users can receive assets through the `receiveAssets` function.
- Users can perform a refill through the `refill` function.
- Users can claim unSettlePnl through the `claimUnSettlePnl` function.

PnlHandler

- The pnlHandler can send assets through the `sendAssets` function.

4 Findings

HVA-1 The Condition Check in `setLiquidationBonus` Is Incorrect

Severity: Medium

Status: Fixed

Code Location:

contracts/hybridvault/HybridVault.sol#146-150

Descriptions:

In the `setLiquidationBonus` function, `_liquidationBonus` is required to be `<PercentageMath.PERCENTAGE_FACTOR`, which is an incorrect condition. `_liquidationBonus` should always be greater than `1e4`, which is `100%`, as set during initialization. This incorrect condition renders the function ineffective.

```
  function setLiquidationBonus(uint _liquidationBonus) external onlyOwner {
    require(_liquidationBonus > 0 && _liquidationBonus <
PercentageMath.PERCENTAGE_FACTOR, "HybridVault: invalid value");
    liquidationBonus = _liquidationBonus;
    emit OwnerSetLiquidationBonus(_liquidationBonus);
  }
```

```
  function initialize(
    address _vUSD,
    address _kToken,
    address _quoteToken,
    address _priceRouter
  ) public initializer {
    __owner_governable_init();
    vUSD = _vUSD;
    kToken = IKToken(_kToken);
    quoteToken = _quoteToken;
    quoteAssetsMinBp = 5000; //50%
    priceRouter = IPriceRouter(_priceRouter);
    PART_WITHDRAW_HF_THRESHOLD = 20000; //200%
    liquidationBonus = 10500; //105%
```

```
        IVUSD(vUSD).approve(_kToken, type(uint).max);
    }
```

Suggestion:

It is recommended to implement the correct condition check.

Resolution:

This issue has been fixed. The client has implemented the correct condition check.

HVA-2 Only the Use of the Pyth oracle is Allowed to Incur a Fee

Severity: Minor

Status: Fixed

Code Location:

contracts/hybridvault/HybridVault.sol#168-209

Descriptions:

Currently, updating prices with the Pyth oracle incurs a fee. If the `msg.value` passed by the user is greater than 0 and the user does not use the Pyth oracle, this amount will be a loss for the user.

```
function priceOfUnderlying(bytes calldata data) public override payable {
    OracleSource source = OracleSource(uint8(data[0]));
    uint tokenId = uint(uint8(data[1]));
    if (source == OracleSource.KILO_SIGNATURE) {
        priceOfSignature(tokenId, data);
    } else if (source == OracleSource.PYTH) {
        (bytes32 pythId, bytes[] memory priceUpdateData) = abi.decode(data[2:], (bytes32,
bytes[]));
        priceOfPyth(tokenId, pythId, priceUpdateData);
    } else if (source == OracleSource.CHAINLINK) {
        (address token) = abi.decode(data[2:], (address));
        priceOfChainLink(tokenId, token);
    } else if (source == OracleSource.KILO_EX) {
        priceOfKiloEx(tokenId, data);
    } else if (source == OracleSource.MOCK_ORACLE) {
        priceOfMock(tokenId, data);
    }
}
```

Suggestion:

It is recommended to check whether the Pyth oracle is used when `msg.value` passed by the user is greater than 0.

Resolution:

This issue has been fixed. The client has checked whether the Pyth oracle is used when `msg.value` passed by the user is greater than 0.

PRO-1 Drain the Native Tokens from the PriceRouter

Severity: Major

Status: Fixed

Code Location:

contracts/hybridvault/PriceRouter.sol#94

Descriptions:

In the `PriceRouter.priceOfPyth()` function, the protocol calls `pyth.updatePriceFeeds()` to update the price, and the fee for updating the price is paid using the protocol's funds.

```
  function priceOfPyth(uint tokenId, bytes32 pythId, bytes[] memory priceUpdateData)
internal returns (uint) {
      require(oracleSources[tokenId] == OracleSource.PYTH, "PriceRouter: not allowed");
      uint fee = pyth.getUpdateFee(priceUpdateData);
      pyth.updatePriceFeeds{ value: fee }(priceUpdateData);
      PythStructs.Price memory priceInfo = pyth.getPriceNoOlderThan(pythId,
maxOldAge);
      uint oPrice = uint(uint64(priceInfo.price));
      uint price;
      if (priceInfo.expo >= 0) {
          uint exponent = uint(uint32(priceInfo.expo));
          price = oPrice * PRICE_BASE * (10 ** exponent);
      } else {
          uint exponent = uint(uint32(-priceInfo.expo));
          price = (oPrice * PRICE_BASE) / (10 ** exponent);
      }
      kiloExPrices[tokenId] = PriceInfo(price, block.timestamp);
      return price;
  }
```

A malicious user can call `HybridVault.deposit()` to deposit 0 or a very small amount of hAssets and repeatedly call `PriceRouter.priceOfPyth()`, depleting the contract's funds by continuously triggering the price update.

Suggestion:

It is recommended to have users pay the fee for updating the oracle.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PRO-2 The Expired Price Check is Incorrect

Severity: Medium

Status: Fixed

Code Location:

contracts/hybridvault/PriceRouter.sol#105;

contracts/hybridvault/PriceRouter.sol#113

Descriptions:

In the `PriceRouter.priceOfPyth()` and `PriceRouter.priceOfChainLink()` functions, the protocol updates `PriceInfo` as `PriceInfo(price, block.timestamp)`, where the current time `block.timestamp` is used instead of the timestamp from when the price was published by the oracle

```
kiloExPrices[tokenId] = PriceInfo(price, block.timestamp);
```

Later, when the protocol calls `priceRouter.getPriceNoOlderThan()` to ensure the price is not outdated, the function checks if `block.timestamp - priceInfo.timestamp <= maxOldAge`. Since `priceInfo.timestamp` is set to `block.timestamp`, this validation will always pass and does not effectively prevent the use of outdated prices.

Suggestion:

It is recommended to set `priceInfo.timestamp` to the timestamp of when the price was published by the oracle. https://github.com/pyth-network/pyth-crosschain/blob/94f1bd54612adc3e186eaf0bb0f1f705880f20a6/target_chains/ethereum/sdk/solidity

```
struct Price {
    // Price
    int64 price;
    // Confidence interval around the price
    uint64 conf;
    // Price exponent
    int32 expo;
    // Unix timestamp describing when the price was published
```

```
    uint publishTime;
}
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PRO-3 Users can Evade Updating the Oracle's Price

Severity: Medium

Status: Fixed

Code Location:

contracts/hybridvault/PriceRouter.sol#92

Descriptions:

In the `priceOfPyth()` function, the protocol calculates the fee based on `priceUpdateData` and then updates the price, allowing a maximum time of 90 seconds.

```
function priceOfPyth(uint tokenId, bytes32 pythId, bytes[] memory priceUpdateData)
internal returns (uint) {
        require(oracleSources[tokenId] == OracleSource.PYTH, "PriceRouter: not allowed");
        uint fee = pyth.getUpdateFee(priceUpdateData);
        pyth.updatePriceFeeds{ value: fee }(priceUpdateData);
        PythStructs.Price memory priceInfo = pyth.getPriceNoOlderThan(pythId,
maxOldAge);
        uint oPrice = uint(uint64(priceInfo.price));
        uint price;
        if (priceInfo.expo >= 0) {
            uint exponent = uint(uint32(priceInfo.expo));
            price = oPrice * PRICE_BASE * (10 ** exponent);
        } else {
            uint exponent = uint(uint32(-priceInfo.expo));
            price = (oPrice * PRICE_BASE) / (10 ** exponent);
        }
        kiloExPrices[tokenId] = PriceInfo(price, block.timestamp);
        return price;
    }
```

A user can evade updating the price by passing empty `priceUpdateData` after each normal transaction.

Suggestion:

It is recommended to check that the length of `priceUpdateData` is greater than 0.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PRO-4 Signature Replay

Severity: Minor

Status: Acknowledged

Code Location:

contracts/hybridvault/PriceRouter.sol#123-135

Descriptions:

In the `priceOfSignature.priceOfSignature()` function, when `priceOfSignatureId[tokenId]` is less than the `timestamp`, the protocol verifies if the signature is from the specified address and then updates `kiloExPrices`. The issue here is that this signature can be replayed.If a malicious user uses an old signature, they will use outdated prices. Even if the market price of the token has increased, this could negatively impact the protocol.

```
    function priceOfSignature(uint tokenId, bytes calldata data) public {
        (uint price, uint timestamp, bytes memory signature) = abi.decode(data[2:], (uint,
uint, bytes));
        if(priceOfSignatureId[tokenId] < timestamp) {
            bytes32 _msgHash = toEthSignedMessageHash(getMessageHash(tokenId, price,
timestamp));
            require(verify(_msgHash, signature), "PriceRouter: Invalid Signer!");
            if (timestamp > block.timestamp) { //input timestamp may be larger than the block
time
                timestamp = block.timestamp;
            }
            kiloExPrices[tokenId] = PriceInfo(price, timestamp);
            priceOfSignatureId[tokenId] = timestamp;
        }
    }
```

Suggestion:

It is recommended to mark used signatures to prevent replay attacks.

PRO-5 Unused `isKeeper` Variable

Severity: Minor

Status: Fixed

Code Location:

contracts/hybridvault/PriceRouter.sol#18

Descriptions:

The `isKeeper` variable is likely intended to manage certain operational permissions, but it is currently not used in any functions. This means its intended purpose is not realized, possibly indicating a missing functionality. Unused variables increase code complexity and negatively impact code readability and maintainability.

```
mapping(address => bool) public isKeeper;
```

Suggestion:

It is recommended to remove it from the code if it is not necessary.

Resolution:

This issue has been fixed. The client has removed it from the code.

VSR-1 Redundant Inheritance

Severity: Informational

Status: Acknowledged

Code Location:

contracts/core/VaultStakeReward.sol#17

Descriptions:

The `VaultStakeReward` contract inherits both `ERC20Upgradeable` and `KiloERC4626Upgradeable`, but since the `KiloERC4626Upgradeable` contract already inherits from `ERC20Upgradeable`, it is unnecessary to continue inheriting from `KiloERC4626Upgradeable` here.

```
// VaultStakeReward.sol
...
contract VaultStakeReward is OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable,
PausableUpgradeable, ERC20Upgradeable, KiloERC4626Upgradeable {
...
// KiloERC4626Upgradeable.sol
...
abstract contract KiloERC4626Upgradeable is Initializable, ERC20Upgradeable,
IERC4626Upgradeable {
...
```

Suggestion:

It is recommended to remove the VaultStakeReward inheritance statement for `ERC20Upgradeabl`.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.