SMART CONTRACTS REVIEW

October 28th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING UNLOCKD

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 2 Critical issues: 2 resolved = 0 points deducted
- 3 High issues: 3 resolved = 0 points deducted
- 19 Medium issues: 12 resolved and 7 acknowledged = - 3 points deducted
- 18 Low issues: 13 resolved and 5 acknowledged = - 1 points deducted
- 23 Informational issues: 17 resolved and 6 acknowledged = 0 points deducted

Thus, 100 - 3 - 1 = 96

TECHNICAL SUMMARY

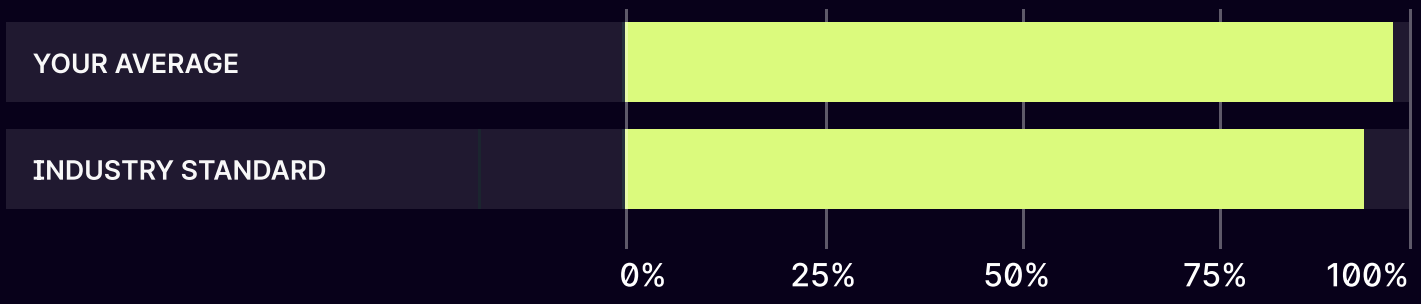
This document outlines the overall security of the Unlockd smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Unlockd smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



98% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Unlockd team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Unlockd repositories:

MaxApy repo: https://github.com/UnlockdFinance/maxapy
Last commit: 8d7470506a081efdda4951e55dfd7805c2615d51

Within the scope of this audit, the team of auditors reviewed the following contract(s):

```
src/
├── MaxApyRouter.sol
├── MaxApyVault.sol
├── MaxApyVaultFactory.sol
├── helpers
│   ├── AddressBook.sol
│   └── VaultTypes.sol
├── interfaces
│   ├── IAlgebraPool.sol
│   ├── IBalancer.sol
│   ├── IBeefyVault.sol
│   ├── ICellar.sol
│   ├── IConvexBooster.sol
│   ├── IConvexRewards.sol
│   ├── ICurve.sol
│   ├── IHypervisor.sol
│   ├── IMaxApyRouter.sol
│   ├── IMaxApyVault.sol
│   ├── IStakingRewardsMulti.sol
│   ├── IStrategy.sol
│   ├── IUniProxy.sol
│   ├── IUniswap.sol
│   ├── IWETH.sol
│   ├── IWrappedToken.sol
│   ├── IWrappedTokenGateway.sol
│   ├── IYVault.sol
│   └── IYVaultV3.sol
```

AUDITING STRATEGY AND TECHNIQUES APPLIED

```
├── lib
│   ├── Constants.sol
│   ├── ERC20.sol
│   ├── FixedPoint96.sol
│   ├── Initializable.sol
│   ├── LiquidityRangePool.sol
│   ├── LiquidityTokenMath.sol
│   ├── OracleLibrary.sol
│   └── ReentrancyGuard.sol
├── periphery
│   └── MaxApyHarvester.sol
└── strategies
    ├── base
    │   ├── BaseBeefyCurveStrategy.sol
    │   ├── BaseBeefyStrategy.sol
    │   ├── BaseConvexStrategy.sol
    │   ├── BaseConvexStrategyPolygon.sol
    │   ├── BaseSommelierStrategy.sol
    │   ├── BaseStrategy.sol
    │   ├── BaseYearnV2Strategy.sol
    │   └── BaseYearnV3Strategy.sol
    ├── mainnet
    │   ├── DAI
    │   │   └── yearn
    │   │       ├── YearnAjnaDAIStakingStrategy.sol
    │   │       └── YearnDAIStrategy.sol
    │   ├── USDC
    │   │   ├── convex
    │   │   │   └── ConvexCrvUSDWethCollateralStrategy.sol
    │   │   ├── sommelier
    │   │   │   └── SommelierTurboGHOStrategy.sol
    │   │   └── yearn
    │   │       ├── YearnLUSDStrategy.sol
    │   │       └── YearnUSDCStrategy.sol
    │   └── WETH
    │       ├── convex
    │       │   └── ConvexdETHFrxETHStrategy.sol
```

AUDITING STRATEGY AND TECHNIQUES APPLIED

```
├── sommelier
│   ├── SommelierMorphoEthMaximizerStrategy.sol
│   ├── SommelierStEthDepositTurboStEthStrategy.sol
│   ├── SommelierTurboDivEthStrategy.sol
│   ├── SommelierTurboEEthV2Strategy.sol
│   ├── SommelierTurboEthXStrategy.sol
│   ├── SommelierTurboEzEthStrategy.sol
│   ├── SommelierTurboRsEthStrategy.sol
│   ├── SommelierTurboStEthStrategy.sol
│   └── SommelierTurboSwEthStrategy.sol
└── yearn
    ├── YearnAaveV3WETHLenderStrategy.sol
    ├── YearnAjnaWETHStakingStrategy.sol
    ├── YearnCompoundV3WETHLenderStrategy.sol
    ├── YearnV3WETH2Strategy.sol
    ├── YearnV3WETHStrategy.sol
    └── YearnWETHStrategy.sol
└── polygon
    └── USDCe
        ├── beefy
        │   ├── BeefyCrvUSDUSDCeStrategy.sol
        │   ├── BeefyMaiUSDCeStrategy.sol
        │   └── BeefyUSDCeDAIStrategy.sol
        ├── convex
        │   ├── ConvexUSDCCrvUSDStrategy.sol
        │   └── ConvexUSDTCrvUSDStrategy.sol
        └── yearn
            ├── YearnAaveV3USDTLenderStrategy.sol
            ├── YearnAjnaUSDCStrategy.sol
            ├── YearnCompoundUSDCeLenderStrategy.sol
            ├── YearnDAILenderStrategy.sol
            ├── YearnDAIStrategy.sol
            ├── YearnMaticUSDCStakingStrategy.sol
            ├── YearnUSDCeLenderStrategy.sol
            ├── YearnUSDCeStrategy.sol
            └── YearnUSDTStrategy.sol
```

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Unlockd repositories:

unlockd repo: https://github.com/UnlockdFinance/unlockd-v2
Last commit: 3bada063d1b7f8bf7ce872902fba001ffddc5479

Within the scope of this audit, the team of auditors reviewed the following contract(s):

IBasicWalletVault.sol
ICryptoPunksMarket.sol
IERC11554K.sol
IERC11554KController.sol
ISablierV2LockupLinear.sol
IUSablierLockupLinear.sol
BaseERC1155Wrapper.sol
BaseERC6960Wrapper.sol
BaseERC721Wrapper.sol
BaseEmergency.sol
- PolytradeAdapter.sol

ReservoirAdapter.sol
SablierAdapter.sol
WrapperAdapter.sol
U4K.sol
UPolytrade.sol
USablierLockupLinear.sol
MaxApy.sol
BasicWalletFactory.sol
BasicWalletRegistry.sol
BasicWalletVault.sol

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Unlockd repositories:

maxStrategist repo: https://github.com/UnlockdFinance/maxStrategist
Last commit: 2ca096845ca64afefea2a1ce693ac276bfe129e5

Within the scope of this audit, the team of auditors reviewed the following contract(s):

```
src/
├── MaxStrategist.sol
└── interfaces
    ├── IERC20.sol
    ├── IERC20Metadata.sol
    ├── IERC4626.sol
    ├── IMaxApyVault.sol
    └── IStrategy.sol
```

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Unlockd smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Testing contract/s logic against common and uncommon attack vectors.

Executive Summary

`MaxApy protocol is a cutting-edge DeFi yield optimization protocol designed to maximize returns on cryptocurrency assets by automating complex process of yield farming ensuring that assets are always allocated in profitable strategies. MaxApy codebase for audit consists Vault contract, Router contract , Harvester contract and all strategies contracts.

MaxApy Vaults are single-asset vaults which allow users to deposit their assets for vault shares. These shares value keep increasing as vaults deploy deposited assets into strategies generating yield. This complete process is automatic and optimized for best yields.

MaxApy Router contract is a helper contract to safely and easily interact with MaxApy vaults. It allows users to deposit and redeem to the vaults. MaxApy Harvester contract is to call harvest in an atomic for multiple strategies.

FInally, there are diverse range of Strategy contracts implementations that integrate with various DeFi protocols. Each strategy is responsible for depositing funds into a specific protocol, monitoring the position, and harvesting profits when appropriate.`

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Unlockd team and the Unlockd team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

MAXAPY

CRITICAL-1 | RESOLVED

Methods depositWithPermit and mintWithPermit can be used to steal users' shares

In Contract MaxApyVaultV2.sol, the method depositWithPermit(address owner,..., address receiver) allows a user to sign the permit txn and anyone to send the deposit txn on behalf of the user.

Any malicious user can see the transaction details in the mempool, and front-run this txn setting his address as the `receiver` address. In this case, assets will be deducted from the `owner` account as both methods call the internal method _deposit() which has the following logic:

```
asset().safeTransferFrom(by, address(this), assets);
```

The minted shares will be sent to the `receiver` account.

```
_mint(to, shares);
```

Recommendation:
Update the methods to use the `owner` address as the `receiver` address as well.

Sommelier cellars' redeem or withdraw method can send multiple tokens to Strategy contracts

Strategies using Sommelier Cellars as an underlying vault are under the risk of receiving multiple tokens when withdrawing assets from Sommelier Cellars.

For eg: Cellar.redeem(...) and Cellar.withdraw(...) being used in multiple strategies contracts

The current implementation of Sommelier Strategy contracts assumes receiving only underlying assets on withdrawing or redeeming. This leads to the possibility of loss of funds and those funds being stuck forever in the strategies contract.

Sommelier Cellar implementation warns about the same here.

Sommelier strategies that are currently sending multiple tokens in redeem:

```
SommelierMorphoEthMaximizerStrategy =>
https://etherscan.io/tx/0x218ced27650a3db085d7c2369dd8d67c4d85466b6ee28057
8fa3631b1cf846e2

SommelierTurboGHOStrategy => USDC or GHO

SommelierTurboEthXStrategy =>
https://etherscan.io/tx/0xe631b475f5b2b64adef2344168eed1dfd005b7c0cdf81356
d5d57ab475df13e2
```

Other Sommelier strategies that have a warning (in contract code) for sending multiple tokens in redeem:

```
SommelierTurboEEthV2Strategy
SommelierTurboEzEthStrategy
SommelierTurboRsEthStrategy
SommelierTurboStEthStrategy
SommelierTurboSwEthStrategy
SommelierStEthDepositTurboStEthStrategy
SommelierTurboDivEthStrategy
```

Note: cellar.redeem returns amount of asset redeemed. Adding swapped withdrawal amount to it not correct.

Recommendation:
Update the Sommelier Strategies logic for the above-mentioned contracts to handle the case when multiple tokens are withdrawn or redeemed by the strategies contracts.

Yearn V3 Strategies can lose funds while liquidating the exact amount

In Contract YearnAjnaDAIStakingStrategy.sol, the method _liquidateExact(...) calculated the amount to withdraw from Yearn V3 vault to allow the withdrawal.
This amount is converted to yearn v3 vault shares. To withdraw the amount needed, firstly yearn vault shares need to be unstaked from yearn staking contract and then burned to withdraw the assets. It is done as follows:

```
uint256 amountToWithdraw = amountNeeded - underlyingBalance;
        uint256 neededVaultShares =
yVault.previewWithdraw(amountNeeded);
        yearnStakingRewards.withdraw(neededVaultShares);
        uint256 burntShares = yVault.withdraw(
            amountToWithdraw,
            address(this),
            address(this)
        );
```

Here, amountToWithdraw is calculated correctly but while calculating the needed yearn v3 vault shares, `amountNeeded` is passed.
Due to this, the withdrawn vault shares will be more than required. It is to be noted thought the vault shares burned only for withdrawing assets for the amount `amountToWithdraw`.
This will lead to yearn v3 vault shares left in the contract. These left shares can not be withdrawn or converted to assets anymore.

Further, method _shareBalance() will provide the wrong amount of yearn vault shares since some shares will be in the strategy contract.

```
  function _shareBalance() internal view override returns (uint256
_balance) {
      return yearnStakingRewards.balanceOf(address(this));
  }
```

This will lead to _estimatedTotalAssets() being calculated wrong every time affecting other calculations such as harvest().

```
function _estimatedTotalAssets()
      internal
      view
      virtual
      override
```

```
    returns (uint256)
{
    return _underlyingBalance() + _shareValue(_shareBalance());
}
```

The same issue exists in YearnAjnaWETHStakingStrategy.sol and YearnMaticUSDCStakingStrategy.sol strategies as well.

Recommendation:
Update the logic to calculate the neededVaultShares correctly.

MEDIUM-1 | RESOLVED

Incorrect `loss` calculation for Base strategy contracts

In Contract BaseSommelierStrategy.sol, the method liquidateExact(...) calculate loss as follows:

```
loss = _sub0(_shareValue(burntShares), amountNeeded);
```

Here, `amountNeeded` will always be greater than burn shares value as shares are burned corresponding to `amountToWithdraw = amountNeeded - underlyingBalance`.

This will lead to `loss` being calculated as 0 all the time leading to further miscalculations.

The same issue is with BaseYearnV3Strategy.sol as well.

Recommendation:
Update the loss calculation as follows:

```
// use sub zero because shares could be fewer than expected and
underflow
        loss = _sub0(_shareValue(burntShares), amountToWithdraw);
```

Incorrect net off for unrealized gain and loss

In Contract BaseSommelierStrategy.sol, the method _prepareReturn(...) calculates the net-off unrealized gain and loss as follows:

```
// Net off unrealized profit and loss
                switch lt(unrealizedProfit, loss)
                // if (unrealizedProfit < loss)

                case true {
                    realizedProfit := 0
                }
                case false {
                    unrealizedProfit := sub(unrealizedProfit, loss)
                    loss := 0
                }
```

Here, if unrealizedProfit < loss, then realizedProfit is set to 0. Instead, it should be

```
case true {
                  loss := sub(loss, unrealizedProfit)
                  unrealizedProfit := 0
                }
```

This miscalculation for unrealizedGain can lead to wrong fee calculation leading to incorrect vault shares minting.

This issue exists in all Base strategies, child strategies of Yearn V3, Sommelier and Convex.

Recommendation:
Update as suggested to correctly net off unrealized gain and loss.

Incorrect `loss` calculation for Yearn v3 strategies

In Contract YearnAjnaDAIStakingStrategy.sol, the method liquidateExact(...) calculate loss as follows:

```
loss = _shareValue(burntShares) - amountNeeded;
```

Here, `amountNeeded` will always be greater than the burned shares value as shares are burned corresponding to `amountToWithdraw = amountNeeded - underlyingBalance`.

This will lead to an underflow issue leading to txn being reverted.

The same issue exists in YearnAjnaWETHStakingStrategy.sol and YearnMaticUSDCStakingStrategy.sol strategies as well.

Recommendation:
Update the loss calculation as follows:

```
  // use sub zero because shares could be fewer than expected and
underflow
            loss = _sub0(_shareValue(burntShares), amountToWithdraw);
```

The SommelierTurboDivEthStrategy strategy divests in terms of underlying assets directly instead of Balancer LP tokens from Cellar

In Contract SommelierTurboDivEthStrategy.sol, the method _prepareReturn has the following logic to divest assets from the underlying sommelier cellar:

```
if (amountToWithdraw > underlyingBalance) {
                uint256 expectedAmountToWithdraw = Math.min(
                    maxSingleTrade,
                    amountToWithdraw - underlyingBalance
                );

                uint256 sharesToWithdraw = (expectedAmountToWithdraw);

                uint256 withdrawn = _divest(sharesToWithdraw);
... }
```

For this strategy, expectedAmountToWithdraw needs to be converted to Balancer LP tokens and then those LP tokens need to be withdrawn from Sommelier Cellar. Further, LP tokens will be used to exit the Balancer pool to finally get WETH.

Since here, expectedAmountToWithdraw is directly used as LP tokens, assets withdrawn will be more than needed. Although extra withdrawn WETH will be invested back, still this scenario should be avoided as it reduces the yield over time.

Recommendation:
Convert the expectedAmountToWithdraw amount to LP tokens before divesting.

Missing transaction deadline checks can result in reward tokens being sold at less price

In Contract YeahAjnaWETHStakingStrategy.so, the method _unwindRewards(...) swaps reward token Ajna to WETH tokens as follows:

```
router.exactInputSingle(
                IRouter.ExactInputSingleParams({
                    tokenIn: ajna,
                    tokenOut: underlyingAsset,
                    fee: 10000,
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: ajnaBalance,
                    amountOutMinimum: 0,
                    sqrtPriceLimitX96: 0
                })
```

Here, the deadline is set as `block.timestamp` which basically disables the transaction expiration check because the deadline will be set to whatever timestamp the block including the transaction is minted at.

Claiming and selling reward tokens can be exploited by a sandwich attack. Depositors may receive less yield than expected due to reward tokens being sold at an outdated price.

A similar issue exists for YearnLUSDStrategy.sol's _invest(...) and _divest() methods and in YearnAjnaDAIStakingStrategy's _unwindRewards(...) and YearnMaticUSDCStakingStrategy's _unwindRewards() method.

Recommendation:
Consider a reasonable value to the deadline argument. For example, Set it to 30 minutes on the Ethereum mainnet. Also, consider letting the admin change the value when on-chain conditions change and may require a different value.

Balancer join pool and exit pool can have potential high slippage as joinPool is not estimating BPT tokens and minAmountsOut[] is an empty array

In Contract SommelierTurboDivEthStrategy.sol, the method _joinPool() joins the Balancer pool as follows:

```
JoinPoolRequest memory joinParams = JoinPoolRequest({
        assets: _assets,
        maxAmountsIn: amountsIn,
        userData: abi.encode(
            JoinKind.EXACT_TOKENS_IN_FOR_BPT_OUT,
            amountsIn,
            0
        ),
        fromInternalBalance: false
    });
```

Here, the expected amount of BPT tokens is set as 0 which is advised in the Balancer doc to be calculated and set when joining the pool to avoid unexpected results. This is how it is advised in Balancer docs for joining the pool with maxAmountsIn.

Similarly, in Contract SommelierTurboDivEthStrategy.sol, the method _exitPool() exits the Balancer pool as follows:

```
uint256[] memory _minAmountsOut = new uint256[](2);

    ExitPoolRequest memory exitRequest = ExitPoolRequest({
        assets: _assets,
        minAmountsOut: _minAmountsOut,
        userData: abi.encode(
            ExitKind.EXACT_BPT_IN_FOR_ONE_TOKEN_OUT,
            _lpTokens,
            1
        ),
        toInternalBalance: false
    });
```

Here, _minAmountOut is for the lower limits for the tokens to receive. In short, what are the minimum amounts you would find acceptable, given the amount of BPT you are providing?

Since _minAmountOut is an empty array so minimum accepted value is 0 and this puts the strategy at risk of accepting withdrawn asset amounts as low as 0. This will cause a huge loss to the protocol if not handled properly.
This is how balancer docs advised when exiting the pool.

Recommendation:
Update the join pool and exit pool methods as suggested in Balancer to achieve results with as little deviation as possible.

Curve pool's exchange() method with max_dy set as 0

In Contract SommelierStEthDepositTurboStEthStrategy.sol, the method _invest(...) and _divest(...) exchange WETH to stETH and stETH to WETH respectively as follows:

```
uint256 stEthReceived = pool.exchange{value: amount}(0, 1, amount, 0);
withdrawn = pool.exchange(1, 0, stEthWithdrawn, 0);
```

As mentioned in Curve docs, the fourth parameter is min_dy which is the expected amount of output tokens from this exchange.

Since these values are passed 0, in invest and divest, there is a risk of receiving as little as 0 output tokens in exchange for input tokens which will be a loss of funds for protocol and users.

This issue also exists in ConvexEthFrxEthStrategy.sol _unwindRewards() method where curve pool is used to swap CVX to WETH. And also in _invest(...) and _divest(...) methods of the same contract where the curve pool is used to swap Eth to frxEth and vice versa.

Recommendation:
Update the exchange method logic by calculating and setting the min_dy value for output tokens. Pool.get_dy(...) method can be used for the same.

Uniswap v3 exactInputSingle(...) executes with amountOutMinimum as `0`

In Contract YearnAjnaWETHStakingStrategy.sol, the method _unwindRewards(..) is swapping rewards Ajna rewards to WETH as following:

```
router.exactInputSingle(
            IRouter.ExactInputSingleParams({
                tokenIn: ajna,
                tokenOut: underlyingAsset,
                fee: 10000,
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: ajnaBalance,
                amountOutMinimum: 0,
                sqrtPriceLimitX96: 0
            })
        );
```

Here, amoutOutMinimum set as 0 means that the transaction will proceed regardless of how much output token you receive. This can be risky as it exposes you to potential slippage and adverse price movements, potentially resulting in receiving far fewer tokens than expected.

A similar issue exists for YearnLUSDStrategy.sol's _invest(...) and _divest(...) methods and in YearnAjnaDAIStakingStrategy's _unwindRewards(...) and YearnMaticUSDCStakingStrategy's _unwindRewards() method.

Recommendation:
Update the amoutOutMinimum value using Uniswap v3's Quoter contract.

Uniswap v3 _estimateAmountMin uses a low time duration of 10 seconds for TWAP calculation

In Contract YearnLUSDStrategy.sol, the method _estimateAmountOut(...) calculates the amount out for output tokens with secondsAgo set as 10 as follows:

```
uint32[] memory secondsAgos = new uint32[](2);

    secondsAgos[0] = secondsAgo;
    secondsAgos[1] = 0;
```

This provides a 10-second window to estimate output amounts for token swaps. Given that the average block time on the Ethereum network is approximately 12 seconds, TWAP with that short duration can be easily influenced by the trades within a single block.

Uniswap v3 pools can be highly volatile and a 10-second window might not capture the price fluctuations.

Recommendation:
Increase a TWAP observation window to at least 1800 seconds that offers a highly stable average, useful for significant trading decisions or low-frequency trading strategies.

Uniswap v2 swapExactTokensForTokens method lacks a proper deadline and amoutOutMin is set to 0

In Contract ETHFrxETHStrategy.so, the method _unwindRewards(...) swaps CRV tokens to WETH using Uniswap v2 as follows:

```
address[] memory path = new address[](2);
          path[0] = crv;
          path[1] = underlyingAsset;
          router.swapExactTokensForTokens(
              crvBalance,
              0,
              path,
              address(this),
              block.timestamp
          );
```

Here, amountOut min is 0 which will let the transaction proceed regardless of how much output token you receive. This can be risky as it exposes you to potential slippage and adverse price movements, potentially resulting in receiving far fewer tokens than expected.

Also, the deadline set is block.timestamp which disables the transaction expiration check because the deadline will be set to whatever timestamp the block including the transaction is minted at.

Claiming and selling reward tokens can be exploited by a sandwich attack. Depositors may receive less yield than expected due to reward tokens being sold at an outdated price.

Recommendation:
Use UniswapV2.getAmountsOut() to estimate the number of output tokens and consider a reasonable value to the deadline argument. For example, Set it to 30 minutes on the Ethereum mainnet.

Yearn v3 withdraw() can fail as no maxLoss specified

In Contract YearnAjnaDAIStakingStrategy.sol, the method liquidateExact(...) executes yVault.withdraw(...) without specifying the maxLoss value. Thus, the Yearn vault's withdraw function will be called with its default maxLoss input value which is 0.01%.

If the total loss incurred during the withdrawal is more than 0.01%, calling the Yearn vault's withdraw function that executes assert totalLoss <= maxLoss * (value + totalLoss) / MAX_BPS will revert.
In a bear market, it is possible that the Yearn vault's strategies do not perform well so the total loss can be more than 0.01% permanently. In this situation, calling the Vault. withdraw or Vault. redeem function will always revert because calling the Yearn vault's withdraw function without specifying the maxLoss input reverts.

This revert can result in the MaxApyVaultv2.withdraw() method failing as well, not letting users withdraw their deposited funds.

The same issue exists in YearnAjnaWETHStakingStrategy.sol and YearnMaticUSDCStakingStrategy.sol strategies as well.

Recommendation:
Updated the above with add input that would be used as the maxLoss input for calling the Yearn vault's withdraw function.

MEDIUM-12 | ACKNOWLEDGED

Strategies with extra rewards need to be harvested before exiting strategy

In Contract MaxApyVaultV2.sol, the method exitStrategy(...) liquidates all the positions, revokes the strategy and removes it from the withdrawal queue as well. It does not harvest() the strategy being exited and that will leave extra rewards (for eg. CRV, CVX in the case of ConvexETHFrxETHStrategy.sol) unclaimed.

This issue is valid for ConvexETHFrxETHStrategy, YearnAjnaDAIStakingStrategy

Recommendation:
Update the exitStrategy(...) method to harvest the rewards before liquidating all the positions. Method _liquidateAllPositions(...) can be used.

LOW-1 | ACKNOWLEDGED

Strategies with Low TVLs

Sommelier strategies that have extremely low TVLs:
`SommelierTurboGHOStrategy`

```
SommelierTurboEthXStrategy
```

Sommelier strategies that have low TVLs:

```
SommelierMorphoEthMaximizerStrategy
SommelierTurboEEthV2Strategy
SommelierTurboEzEthStrategy
SommelierTurboRsEthStrategy
SommelierTurboSwEthStrategy
```

Recommendation:
Recheck the TVLs of strategies and take appropriate actions.

LOW-2 | RESOLVED

MaxDeposit may revert due to underflow

In Contract MaxApyVaultV2.sol, the method maxDeposit() returns the maximum amount of assets limit available to deposit to the vault. There is a possibility that it might revert when `totalAssets()` > `depositLimit`. It should return `0` in this case.

Recommendation:
Update the logic to use _sub0(...) to handle the underflow scenario.

LOW-3 | RESOLVED

Method _redeem and _withdraw can be DOSed

In Contract MaxApyVaultV2.sol, the internal methods _redeem and _withdraw have the following logic:

```
uint256 preBalance = underlying.balanceOf(address(this));
uint256 loss = IStrategy(strategy).liquidateExact(
    amountRequested
);

uint256 withdrawn = underlying.balanceOf(address(this)) -
    preBalance;

if (withdrawn == 0) continue;
```

Here, if there are any strategies in the withdrawalQueue[] that withdraw `0` amount of assets, for loop will continue with the same `i` meaning the same strategy again unless all gas is consumed since `i` is not incremented.

Recommendation:
Update the logic as follows:

```
if (withdrawn == 0) {
    unchecked {
        ++i;
    }
    continue;
}
```

LOW-4 | RESOLVED

Role removed twice in method exitStrategy()

In Contract MaxApyVaultV2.sol, the method exitStrategy(...) is removing roles twice on line#1860 and line#1869.

Recommendation:
Remove the duplicate line to save some gas.

LOW-5 | RESOLVED

Recipient can be address(0)

In Contract MaxApyRouter.sol, the method redeemNative(...) has a parameter `recipient` which is not validated if it is address(0) or not. Later on, ETH is sent to the `recipient` address as follows:

```
// Transfer native token back to user
            if iszero(call(gas(), recipient, amountOut, 0x00, 0x00, 0x00,
0x00)) {                // If call failed, throw the
`FailedNativeTransfer()` error
                mstore(0x00, 0x3c3f4130)
                revert(0x1c, 0x04)
            }
```

This could lead to loss of funds if the `recipient` address is accidentally set as address(0).

Recommendation:
Add the check to ensure the recipient's address is not address(0).

Harvester may not receive harvest fee if the strategy is in emergency exit mode

In Contract BaseStrategy.sol, the method harvest(...) checks if the harvester address is not address(0) then it should be set as managementFeeReceiver. But it also checks if the contract is in emergency exit mode.

If the contract is in emergency exit mode, the strategy will liquidate all positions. It will report back the vault using the report() method but unrealisedGain will be 0.

Vault uses unrealized gain to calculate the management fee for the user who initiated the harvest in the _assesFee() method. Since unrealisedGain is 0 if the strategy is in emergency mode, the management for the user will be 0 as well.

This will lead to a loss of funds for the user in terms of gas money which was supposed to be refunded in the form of the management fee.

Recommendation:
Although it is the responsibility of the admins to set the strategy in emergency exit mode and call the harvest() method, it is advised to put a check to revert the transaction in case a user calls the harvest in emergency exit mode.

Sommelier strategies not claiming SOMM and other rewards

Few sommelier strategies reward SOMM tokens for depositing to those cellars. These SOMM tokens are not claimed and hence not added to the yield as well.

Apart from SOMM tokens, other reward points/tokens are rewarded by various sommelier strategies as mentioned below:

```
SommelierMorphoEthMaximizerStrategy => SOMM tokens

SommelierTurboGHOStrategy => SOMM tokens

SommelierTurboEEthV2Strategy => SOMM tokens, Ether.fi, Eigen layer points
SommelierTurboEthXStrategy => SOMM tokens
SommelierTurboEzEthStrategy => SOMM tokens
SommelierTurboRsEthStrategy => SOMM tokens
SommelierTurboSwEthStrategy => SOMM tokens and Pear rewards
SommelierTurboDivEthStrategy => DIVA tokens

YearnV3WETHStrategy => dYFI tokens
```

Recommendation:
Update the _prepareReturn method for these strategies to unwind these rewards as well.

Incorrect maxDeposit() check for SommelierTurboDivEthStrategy

In Contract SommelierTurboDivEthStrategy.sol, the method _invest(...) checks the maximum allowed deposit for the underlying Cellar. This max deposit amount is in Balancer LP tokens, the following check is incorrect:

```
uint256 maxDeposit = cellar.maxDeposit(address(this));
amount = Math.min(amount, maxDeposit);
```

Here, the amount is in WETH and the maxDeposit value is for Balancer LP tokens.

Note: Instead of removing the max deposit check, it can be done once LP tokens are received.

Recommendation:
Convert the amount to LP tokens for checking the max deposit.

LOW-9 | RESOLVED

Strategies invest without checking maxDeposit() limit

In BaseSommelierStrategy.sol, the method _invest(...) deposits assets into cellars without checking if the deposited amount > cellar.maxDeposit().

In this case, the deposit() will revert if an amount more than max deposit limit is being deposited.

A similar issue exists in BaseYearnV2Strategy.sol, BaseYearnV3Strategy.sol, YearnAjnaDAIStakingStrategy, YearnAjnaWETHStakingStrategy, YeanMaticUSDCStakingStrategy, YearnLUSDStrategy.sol as well.

Recommendation:
Update the _invest(...) to add this check as follows before investing:

```
uint256 maxDeposit = cellar.maxDeposit(address(this));
 amount = Math.min(amount, maxDeposit);
```

INFORMATIONAL-1 | RESOLVED

Set calldata size is more than required

In Contract MaxApyRouter.sol, the method depositNative(...) wraps ETH to WETH as follows:

```
if iszero(
            call(
                gas(), // Remaining amount of gas
                cachedWrappedToken, // Address of `wrappedToken`
                callvalue(), // `msg.value`
                0x1c, // byte offset in memory where calldata starts
                0x24, // size of the calldata to copy
                0x00, // byte offset in memory to store the return data
                0x00 // size of the return data
            )
```

Here, the size of the calldata to copy should be `0×04` rather than `0×24`.

Recommendation:
Update the size of the calldata to `0×04`.

INFORMATIONAL-2 | RESOLVED

Use unchecked {...} for saving gas

In Contract BaseSommelierStrategy, the method liquidateExact(...) calculates amountToWithdraw as follows:

```
uint256 amountToWithdraw = amountNeeded - underlyingBalance;
```

Given that amountNeeded > underlyingBalance already, unchecked {...} can be used to save gas.

Recommendation:
Update as suggested to save gas.

INFORMATIONAL-3 | ACKNOWLEDGED

Use efficient address(0) check

In Contract BaseStrategy.sol, the init method checks if _strategist is address(0) or not as follows:

```
if eq(_strategist, 0) {..}
```

There is a more efficient way to do the same following:

```
if iszero(shl(96, to)) {..}
```

Recommendation:
Use the suggested method to save gas.

maxSingleTrade and minSingleTrade values not as per assets

In Contract YearnAjnaDAIStakingStrategy.sol, the variable minSingleTrade is set as 1e4 which is quite low considering DAI token's decimals value is 18.

In Contract YearnMaticUSDCStakingStrategy.sol, the variable maxSingleTrade is set as `1000e18` which is a very significant value for USDC asset which has a decimal value of 6.

Recommendation:
Consider updating the minSingleTrade and maxSingleTrade values as per asset decimals.

Yearn v3 vaults staking rewards can be in multiple tokens

In Contract YearnAjnaDAIStakingStrategy.sol, the method _unwindRewards(...) get rewards from the yearn staking pool as follows:

```
// Claim Ajna rewards
        _yearnStakingRewards.getReward();
```

Here, when we check the code for the method getReward():

```
  function getReward() public nonReentrant updateReward(msg.sender) {
        for (uint i; i < rewardTokens.length; i++) {
            address _rewardsToken = rewardTokens[i];
            uint256 reward = rewards[msg.sender][_rewardsToken];
            if (reward > 0) {
                rewards[msg.sender][_rewardsToken] = 0;
                IERC20(_rewardsToken).safeTransfer(msg.sender, reward);
                emit RewardPaid(msg.sender, _rewardsToken, reward);
            }
        }
    }
```

There seems to be the possibility of rewards being sent in multiple tokens but the current logic of the method _unwindRewards handle only 1 reward token.

This will lead to loss of rewards funds eventually affecting strategy total yield.

The same issue exists in YearnAjnaWETHStakingStrategy.sol and YearnMaticUSDCStakingStrategy.sol strategies as well.

Recommendation:
Consider handling the scenario by swapping all reward tokens to the underlying asset.

Method liquidateExact transfers fund to msg.sender instead of vault address

In several strategies, the method liquidateExact sends the underlying assets to the `msg.sender` as follows:

```
nderlyingAsset.safeTransfer(msg.sender, amountNeeded);
```

As this method can be called only by role `VAULT_ROLE`, it is expected that msg.sender will always be the vault contract only but this is just a role and can be assigned mistakenly or maliciously to any address which is not a vault. In that case, assets will be sent to the msg.sender which can not be vault. This is extremely unlikely though.

Also in a few strategies, in the method liquidateExact, the following is checked and if it is not true then the transcation reverts.

```
nderlyingAsset.safeTransfer(msg.sender, amountNeeded);
```

ut the same is not checked in all strategies which seems inconsistent.

Recommendation:
It is advised to update the transfer as follows:

```
underlyingAsset.safeTransfer(vault, amountNeeded);
```

Where `vault` corresponds to

```
IMaxApyVaultV2 public vault;
```

Also, update the liquidateExact to put the check mentioned above in all strategies for consistency.

Wrong comments/Typos

- In Contract MaxApyVaultV2.sol, there is a variable name typo: `uint8 public nexHarvestStrategyIndex;. It should be `nextHarvestStrategyIndex`
- In Contract MaxApyVaultV2.sol, line#396 has a typo `Overflwow`
- In Contract MaxApyVaultV2.sol, line#1158 has an incorrect comment.

```
// Get totalAssets, same as calling _totalAssets() but caching
totalIdle
```

Here, it should be:

```
// Get totalDeposits, same as calling _totalDeposit() but caching
totalIdle
```

In Contract MaxApyVaultV2.sol, there is a typo on line#1363. Correct `shares` and `realising`

* In Contract MaxApyVaultV2.sol, there is an incorrect comment on line#1820. It should be

```
// if (strategies[strategy].strategyDebtRatio == 0)
```

- In Contract BaseStrategy.sol, there is a wrong comment on line#285.
- In Contract SommelierTurboDivEthStrategy.so, there is a wrong comment on line#521. It should be `Balancer LP tokens`, not Convex LP tokens.
- In Contract BaseYearnV2Strategy.sol, there is a wrong comment on line#10. It should be `MaxApy YearnV2 strategies`.
- In Contract YearnAjnaDAIStakingStrategy.sol, there is a wrong comment on line#330. It should be `Exchange Ajna <> DAI`.
- In Contract MaxApyRouter.sol, there is a wrong comment on line#346. It should be `Address of `weth`.

Recommendation:

Correct the typos and incorrect statements.

Missing natspec comments

MaxAPYVaultV2.sol ⇒ method report(...) has natspec comments missing for a few arguments.

```
YearnMaticUSDCStakingStrategy => initialize() method
YearnAjnaWETHStakingStrategy => initialize() method
```

Recommendation:
Add/update the natspec comments.

LOW-1 | RESOLVED

Not all allocators granted the ALLOCATOR_ROLE role

In Contract MaxApyHarvester.sol, roles (KEEPER_ROLE and ALLOCATOR_ROLE) are assigned to an array of addresses provided as arguments in the constructor.

For the same, for loop has been used but `length` for iteration on the arrays is the same for keepers array and allocators array.

```
uint256 length = keepers.length;

      // Iterate through each keeper in the array in order to grant
roles.
      for (uint256 i = 0; i < length;) {
          _grantRoles(keepers[i], KEEPER_ROLE);

          unchecked {
              ++i;
          }
      }
      // Iterate through each allocator in the array in order to grant
roles.
      for (uint256 i = 0; i < length;) {
          _grantRoles(allocators[i], ALLOCATOR_ROLE);

          unchecked {
              ++i;
          }
      }
```

This can cause issues when:
keepers.length > allocators. length ⇒ Txn will revert, Contract deployment failed.
keepers.length < allocators.length ⇒ Not all addresses in the allocator's array will be granted ALLOCATOR_ROLE

Recommendation:
Update the `length` variable before the allocator roles assignment `for loop`.

INFORMATIONAL-1 | ACKNOWLEDGED

Harvester contract needs ADMIN_ROLE for all Vaults and KEEPER_ROLE for all strategies

Since the Harvester contract calls method vault.updateStrategyData(...), it needs ADMIN_ROLE access for all the vaults. There are no direct issues since the Harvester contract limits the usage of the role but it is advised to revoke ADMIN_ROLE when the Harvester contract is no longer being used.

Similarly, the Harvester contract will need KEEPER_ROLE for all the strategies and needs to be revoked once there is no usage to avoid any future malicious activities.

Recommendation:
Revoke ADMIN_ROLE for all vaults and KEEPER_ROLE for all strategies for the Harvester contract when it is replaced, out-of-date, or vulnerable.

INFORMATIONAL-2 | RESOLVED

Wrong comments/Typos

Line#61, mentions cryptopunks

Recommendation:
Update the comment on Line#61.

MEDIUM-1 | RESOLVED

Initialized internal variables will not have the same value in the Proxy contract

In Contracts BaseERC1155Wrapper and BaseERC6960Wrapper, the following internal variable is initialized:

```
uint256 internal _counter = 1;
```

Since these contracts are upgradeable and storage will be done in Proxy contracts after deployment, _counter will have a default value of `0` instead of 1 since _counter is initialized in the implementation contract but not in the proxy contract.

This will lead to wrapper NFTs being minted starting with TokenId as `0` which can further lead to unexpected results.

Recommendation:
Set the _counter variable as 1 in the initialize method for both base contracts.

MEDIUM-2 | RESOLVED

Use safeTransferFrom instead of transferFrom for transferring NFTs

In Contract BasicWalletVault, the method withdrawAssets(...) transfers NFTs to address `to`:

```
(success, ) = contractAddress.call(
abi.encodeWithSignature('transferFrom(address,address,uint256)',
address(this), to, value)
        );
```

Here, transferFrom is used to send NFTs but in case the `to` address is a contract that does not implement onERC721Receiver, the NFTs can be stuck forever leading to loss of funds for the users.

Recommendation:
Use safeTransferFrom for transferring NFTs.

MEDIUM-3 | RESOLVED

Method _execTransaction does not reset oneTimeDelegation to false

In Contract BaseWalletVault.sol, the method execTransaction(...) allows any one-time delegator to call any external contract with any method.

This one-time delegation can be used for malicious activities if not reset to false every time the user calls the execTransaction method. This can even lead to reentrancy issues as a one-time delegator can keep calling since _rawExec is a low-level call and data can be passed empty to reenter the BaseWalletVault.

Recommendation:
Reset the mapping oneTimeDelegation for msg.sender to false in the above-mentioned method.

```
  oneTimeDelegation[msg.sender] = false;
```

LOW-1 | RESOLVED

Mismatched Array length

The methods onERC1155BatchReceived(...) and onDLTBatchReceived(...) have array parameters but their lengths are not checked.

The method onERC1155BatchReceived(...) has arrays tokenIds[] and values[] which should be of the same length but it is not validated.

The method onDLTBatchReceived(...) has arrays mainIds[], subIds[] and amounts[] which should be of the same length but it is not validated.

Recommendation:
Validate that the arrays should be of the same length for the above-mentioned methods.

LOW-2 | ACKNOWLEDGED

The direct transfer of NFT to the Wrapper contract would lock NFT forever

In the Base Wrappers contract, methods onERC721Received(...)/onERC115Received(...)/ onERC1155BatchReceived/onDLTReceived/onDLTBatchReceived will mint wrapped NFT only if safeTransferFrom method is used. If a user calls the transfer() method to send an NFT, the NFT will be locked forever in the Wrapper contract and there will be no wrapper NFT minted for the user.

Recommendation:
Warn users to never use the transfer method to directly transfer their NFT RWAs.

LOW-3 | RESOLVED

Delete tokenIds[tokenId] once tokens are burned in the _burn method

The method _baseBurn(...) burns the NFT but does not delete the set tokenIds[tokenId] in BaseERC1155. Since these values are not deleted, the following methods will continue using the mapping:

```
unction tokenURI(uint256 tokenId) public view

function wrappedTokenId(uint256 tokenId) external view
```

Recommendation:
Delete the tokenIds[tokenId] in the _burn method.

LOW-4 | ACKNOWLEDGED

Parameter _guard is not checked for address(0)

The method setWallet(...) in Contract BasicWalletRegistry.sol sets the various parameters for the contract after validation of those parameters except _guard param.

Since _guard is used in wallet creation as well, it is advised to validate that the same is not address(0).

Recommendation:
Validate _that the guard is not address(0).

INFORMATIONAL-1 | RESOLVED

Use onlyInitializing modifier instead of the initializer modifier

In the Base Wrapper contract, all the init methods use an initializer modifier although these contracts are abstract. It is advised to use the onlyInitializing modifier as it is recommended by OpenZeppelin to use the same.

Recommendation:
Use the onlyInitializing modifier for init methods of all Base Wrapper contracts.

INFORMATIONAL-2 | RESOLVED

Wrong comments/Typos/Natspec comments

In BaseERC1155Wrapper.sol, line#78 has the wrong comment as it mentioned Emergency role is needed but the modifier is for the wrapper adapter. This is valid for BaseERC6960Wrapper.sol as well.

In BaseERC1155Wrapper.sol, line#82 reverts with the wrong error, EmergencyAccessDenied whereas it should be NotWrapperAdapter. This is valid for BaseERC6960Wrapper.sol as well.

In U4K.sol, there are multiple typos, line#73, where the comment mentions Sablier protocol but the contract is for 4K protocol.

In BasicWalletRegistry.sol, line#39 needs to add owner as well as owner is allowed as well for the modifier.

In BasicWalletVault.sol, line#10, line#23, line#28, line#143 has typos.

Recommendation:
Update the typos/wrong comments/natspec comments as mentioned above.

INFORMATIONAL-3 | RESOLVED

NewWallet not checked for address(0) and uses Operator as new wallet address

The methods onERC1155BatchReceived(...) and onDLTBatchReceived(...) do not check if the newAddress from the data parameter is address(0) or not.

```
address newWallet = abi.decode(data, (address));
```

Recommendation:
Validate if newWallet is address(0) or not and use operator as the new wallet address.

```
    if (newWallet == address(0)) newWallet = operator;
```

INFORMATIONAL-4 | RESOLVED

`to` address is not ensured as Contract in _rawExec method

The method _rawExec in BaseERC1155 and BaseERC6960 has the following comment:

```
  // Ensure the target is a contract
```

But there is no check to ensure the same.

Recommendation:
It is advised to use the AddressUpgradeable.isContract(...) method for the same.

INFORMATIONAL-5 | RESOLVED

Not disabling the initializer for the implementation contract

In Contract BasicWalletVault, the constructor doesn't call the _disableInitializer() method to disable the implementation contract from being initialized after the contract is deployed.

Recommendation:
Disable the initializer as recommended by OpenZeppelin here

INFORMATIONAL-6 | RESOLVED

No need for return type in method _transferAsset

In Contract BaseWalletvault, the internal method _transferAsset has a return type bool. The method does not return any boolean value.

Recommendation:
It is advised to remove the unused return type if not needed.

UTokenVault emits deposit/withdraw events even when no amount is deposited or redeemed

In Contract UTokenVault, the method deposit and withdraw calls ReserveLogic.strategyDeposit and ReserveLogic.strategyRedeem respectively.

The method ReserveLogic.strategyDeposit has the following logic:

```
if (amountToInvest > 0) { …
  }
```

The method ReserveLogic.strategyRedeem has the following logic:

```
if (amountNeed > 0) { …   }
```

In case amountToInvest == 0 and/or amountNeed == 0, then there wont be any amount deposited or redeemed. Even in that case, UTokenVault will emit events

```
mit Deposit(msg.sender, onBehalfOf, reserve.underlyingAsset, amount);

mit Withdraw(msg.sender, to, reserve.underlyingAsset, amount);
```

Recommendation:
Update the logic to emit events with correct data when needed.

HIGH-1 | RESOLVED

Invest and divest reverts for ConvexUSDCCrvUSD Strategy and ConvexUSDTCrvUSD strategy when a new router is set

In Contract ConvexUSDCCrvUSDStrategy.sol and Contract ConvexUSDTCrvUSDStrategy.sol, method setRouter allows setting a new router for the contract. Although only admin can call this method, setting new router will cause issues as there is no approval for USDC and USDCe for new router.

Strategy swaps USDCe to USDC in _invest method and USDC to USDCe in divest. Without any approval, these methods will fails leading to user funds stuck forever.

Further crvUSD is not approved as well which is needed to swap crvUSD rewards to USDCe. There is a repetition of CRV approvals for the new router.

Recommendation:
Update the setRouter method to do proper approvals. Set approval for the old router to `0` and as needed for the new router for all required tokens.

Incorrect crvUSD to USDCe swap in ConvexUSDTCrvUSD strategy

In Contract ConvexUSDTCrvUSDStrategy.sol, the method _unwindRewards swap crvUSD to USDCe as follows:

```
uint256 crvUsdBalance = _crvUsdBalance();
        if (crvUsdBalance > minSwapCrv) {
            uint256 amountUSDT = curveLpPool.exchange(1, 0, crvUsdBalance,
0);
            zapper.exchange_underlying(2, 1, amountUSDT, 0, address(this));
        }
```

Here, curve pool has 2 coins where index `0` is crvUSD and index 1 is USDT. This logic is swapping index 1 to index `0` which is swapping USDT to crvUSD which is incorrect. Due to this crvUSD will not be swapped to USDT and further not to USDCe as well leading to crvUSD being stuck in the contract.

Recommendation:

Update the logic to swap correctly from crvUSD to USDT.

```
            uint256 amountUSDT = curveLpPool.exchange(0, 1, crvUsdBalance,
0);
```

Unnecessary approvals can lead to loss of funds

For Convex and Yearn strategies, BaseStrategy contracts approves underlying asset to the vaults even if that is not required.

For example: BaseStrategy contract approves USDCe to YearnDAIStrategy and YearnDAILenderStrategy although DAI is deposited to these strategies.

Similarly, USDCe is approved to BeefyStrategy where strategy deposits Curve lp tokens. (Line#92, BaseBeefyStrategy.sol)

This can cause loss of funds for the strategy if any remaining USDCe is transferred into these vaults arbitrarilv.

Recommendation:
Remove any unnecessary approvals in BaseStrategy.sol and initialize() method of strategies contract.

LOW-1 | RESOLVED

Incorrect MaxDeposit check for YearnDAIStrategy and YearnDAILenderStrategy

In Contracts YearnDAIStrategy.sol and YearnDAILenderStrategy.sol, the method _invest(...) checks for maxDeposit of the strategy vault as follows:

```
uint256 maxDeposit = yVault.maxDeposit(address(this));
amount = Math.min(Math.min(amount, maxDeposit), maxSingleTrade);
```

Here, yVault.maxDeposit(...) will return value in 1e18 while maxSingleTrade and amount will be in 1e6 as USDC(e) is the underlying asset.

This will lead to incorrect result as minimum of amount or maxSingleTrade will be selected even if maxDeposit is less than amount.

Recommendation:
Update the logic to account for differences in decimals.

LOW-2 | RESOLVED

Large maxSingleTrade value set

In Contract BaseBeefyStrategy, maxSingleTrade is set as (uint).max which is not re-initialized in BeefyMaiUSDCe strategy leading to investment in BeefyMaiUSDCe to be as large as possible.

Also, ensure maxSingleTrade value being set using the method setMaxSingleTrade is more than minSingleTrade by adding a check.

Recommendation:
Reinitialize maxSingleTrade in the initialize() method in BeefyMaiUSDCe strategy contract.

LOW-3 | RESOLVED

Missing maxDeposit check for ConvexUSDCCrvUSD Strategy and ConvexUSDTCrvUSD strategy

In Contract ConvexUSDCCrvUSDStrategy.sol and Contract ConvexUSDTCrvUSDStrategy.sol, maxDeposit is not used in the _invest method. Invest could revert in case investment amount greater than maxDeposit of the vault.

Recommendation:
Update the logic to check for maxDeposit an deposit minimum of amount and maxDeposit.

INFORMATIONAL-1 | ACKNOWLEDGED

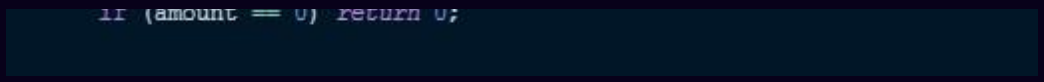
Cvx rewards are not swapped to the underlying asset

In Contract ConvexUSDCCrvUSDStrategy.sol and Contract ConvexUSDTCrvUSDStrategy.sol, method _unwindRewards swaps crv and crvUSD to USDCe but CVX rewards are not swapped. Although currently CVX rewards are `0`, it is advised to swap in case non-zero CVX rewards are present.

Recommendation:
Swap CVX rewards to USDCe.

Unnecessary conditional in method _invest of BeefyStrategy

In BeefyMaiUSDCe strategy, the method, _invest checks on line#70 if the amount > 0 where line#65 has the logic as follows:



So line#70 will always be true.

Recommendation:
Remove the unnecessary check.

Wrong comments/Missing natspec/Typos

In AddressBook.sol, CRV_USD_POLYGON and CRVUSD_POLYGON points to same address. Removed unnecessary vars.
In AddressBook.sol, WMATIC can be renamed to WPOL since MATIC has been renamed to POL.

Missing natspec for param strategist in the initialize() method for all Yearn, Convex and Beefy strategies.

Convex strategies (ConvexUSDCCrvUSD, ConvexUSDTCrvUSD) have a typo on line#39. It mentions Polygon's WETH but address is for Polygon's crvUSD.

Please put the triCryptoPool address (in ConvexUSDCCrvUSD) in AddressBook.sol for consistency.

Line#156 in ConvexUSDCCrvUSD.sol and Line#156 in ConvexUSDTCrvUSD.sol has a wrong comment mentioning about ETH liquidity.

Line#266 in ConvexUSDCCrvUSD.sol has a wrong comment as it mentions swapping CRV to USDC.

Line#350 in ConvexUSDCCrvUSD.sol has a wrong comment as it mentions Convex LP token instead of Curve LP token.

Line#253 in ConvexUSDTCrvUSD.sol has a wrong comment as it mentions fees as `0.005%` instead of `0.05%`.

MEDIUM-1 | RESOLVED

Remove Initializable import

In Contract MaxStrategist.sol, there is an import of Initializable contract which is inherited as well.

```
contract MaxStrategist is Initializable, OwnableRoles {...}
```

The purpose of using initializable import is to make a contract upgradeable and an upgradeable contract needs to have intialize() method with the onlyInitializer modifier. On the contrary, this contract has a constructor that will not initialize the state variables for the proxy contract leading to deployed contract being unusable.

Recommendation:
Remove the initializable import completely as the contract does not need to be upgradeable.

Harvester address in batchRemoveStrategies is not used

In Contract MaxStrategist.sol, the method batchRemoveStrategies(...) accepts a parameter of type HarvestData as follows:

```
function batchRemoveStrategies(
        IMaxApyVault vault,
        HarvestData[] calldata harvests
    ) external checkRoles(KEEPER_ROLE) {.. }
```

Here, HarvestData is defined as follows:

```
struct HarvestData {
        address strategyAddress;
        address harvester;
        uint256 minExpectedBalance;
        uint256 minOutputAfterInvestment;
        uint256 deadline;
    }
```

Further, the harvester address is passed in the strategy.harvest(...) method as follows:

```
strategy.harvest(
                harvests[i].minExpectedBalance,
                harvests[i].minOutputAfterInvestment,
                harvests[i].harvester,
                harvests[i].deadline
            );
```

When we look at the harvest method logic in BaseStrategy, we see the following:

```
function harvest(
        _
        address harvester,
        _
    )
        external
        checkRoles(KEEPER_ROLE)
    {
        // if the harvest was done from the vault means it the
        // harvest was triggered on a deposit
        if (msg.sender == address(vault)) {
            // the depositing user will get the management fees as a reward
            // for paying gas costs of harvest
            managementFeeReceiver = harvester;
        }...}
```

Here managementFeeReceiver is allocated to harvester only if msg.sender for this call is MaxApyVault but in this case, it will be MaxStrategist contract.

This will lead to managementFeeReceiver always being set as address(0) and harvester will never get the management fee.

Recommendation:
Either we pass the harvester as address(0) as we do in MaxApyHarvester contract or we modify the vault to correctly assign the harvester address.

INFORMATIONAL-1 | RESOLVED

Wrong comments/Missing Natspec

Line#8 is an unnecessary import.
Line#73 has a wrong comment.
Lines #25-29 explain HarvestData struct instead of StratData struct.
Lines#134-136 describe about batchAddStrategies method instead of batchRemoveStrategies.

HIGH-1 | RESOLVED

Wrong share value calculation

Strategy BeefyUSDCeDAI calculates beefy shares value in the method _shareVaule(...). First liquidity is calculated based on gamma lp shares and then the amount of tokens (dai, usdce) is calculated based on the liquidity as follows:

```
function _CalcBurnLiquidity(
        int24 tickLower,
        int24 tickUpper,
        uint128 liquidity
    )
        public
        view
        returns (uint256 amount0, uint256 amount1)
    {
        // Pool.burn
        (uint160 sqrtRatioX96, int24 globalTick,,,,,) =
IAlgebraPool(ALGEBRA_POOL).globalState();
        (int256 amount0Int, int256 amount1Int,) =
LiquidityRangePool.computeTokenAmountsForLiquidity(
            tickLower, tickUpper, int128(liquidity), globalTick,
sqrtRatioX96
        );

        amount0 = uint256(amount0Int);

        amount1 = uint256(amount1Int);

        amount0 = _uint128Safe(amount0);

        amount1 = _uint128Safe(amount1);
```

```
        //Pool.collect
        (, uint128 positionFees0, uint128 positionFees1) =
getPositionInfo(tickLower, tickUpper); //@audit position info does not
include burned liquidity amount

        if (positionFees0 > 0 && amount0 > positionFees0) {
            amount0 = positionFees0;
        }

        if (positionFees1 > 0 && amount1 > positionFees1) {
            amount1 = positionFees1;
        }
    }
```

Here, the final amount0 and amount1 are just the positionFees and this does not include the actual burned liquidity tokens amount.

This will significantly affect the share calculation price which will lead to wrong invest and divest leading to loss of funds.

Recommendation:
Add the burned liquidity amounts to the position fees for the correct share value calculation.

LOW-1 | RESOLVED

Unnecessary approval to Uniproxy contract

Strategy BeefyUSDCeDAI approves uint.max USDCe tokens to Uniproxy contract in initialize() method on line#81, which is not needed for the strategy to work.

Recommendation:
Remove the unnecessary approval.

INFORMATIONAL-1 | RESOLVED

Use _uint128Safe

In BeefyUSDCeDAIStrategy, the method computeLiquidityFromShares(...) returns as follows:

```
        return uint128(uint256(position) * shares /
ypervisor.totalSupply()); //@audit use uintsafe
```

Here, uint256 is typecast to uint128 without checking if it overflows.

Recommendation:
Use _uint128Safe as follows:
return _uint128Safe(uint256(position) * (shares) / hypervisor.(totalSupply()));

Check reentrancy lock to prevent read-only reentrancy attack

Contract BeefyUSDCeDAI has a method getPositionInfo which has the logic as follows:

```
(liquidity,,,, tokensOwed0, tokensOwed1) =
IAlgebraPool(ALGEBRA_POOL).positions(positionKey); // @audit
https://docs.algebra.finance/algebra-integral-documentation/algebra-integr
al-technical-reference/integration-process/specification-and-api-of-contra
cts/algebra-pool#positions
```

Algebra pool docs specify that while getting users' position data using the position key, read-only reentrancy attack is possible. [https://docs.algebra.finance/algebra-integral-documentation/algebra-integral-technical-reference/integration-process/specification-and-api-of-contracts/algebra-pool#positions]

Recommendation:
Check the reentrancy lock to prevent the same.

	Contracts
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Unlockd team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Unlockd team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo