SMART CONTRACTS REVIEW

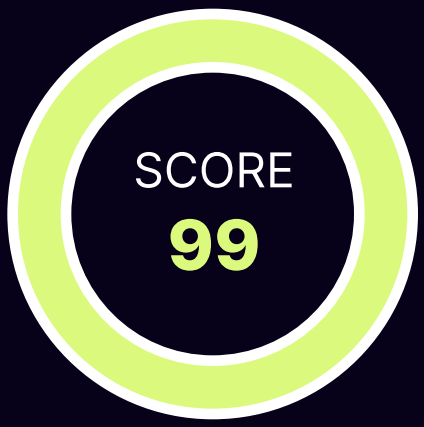
zokyo

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 MAXAPY

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:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 3 Medium issues: 3 resolved = 0 points deducted
- 4 Low issues: 2 resolved and 2 acknowledged = - 1 points deducted
- 7 Informational issues: 6 resolved and 1 acknowledged = 0 points deducted

Thus, 100 - 1 = 99

TECHNICAL SUMMARY

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

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

Contract Status

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

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 maxAPY 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 maxAPY repositories:

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

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 maxAPY 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

Unlockd is the only permissionless RWA liquidity protocol where users can participate as depositors or borrowers. It facilitates the safest, most secure, and cost-effective way to borrow against “tokenized Real-World Assets and financial assets” with instant loans. Unlockd codebase for audit consists of wrappers, wrapper adapters, base wallet and MaxApy strategy integration.

Wrapper contracts are generic wrapper for RWAs that needs to be managed on Unlockd protocol. Along with base wrappers for ERC721, ERC1155, ERC6960, there are specific wrappers implementation of 4K, Polytrade, and Sablier protocols.

Wrapper adapters contracts allows to interact with specific protocols and provides functions for selling and check presell conditions.

Base wallet consists of factory, registry and wallet contract. Factory contracts allows to create a wallet for an owner and registering the same in Registry contract.
MaxApyStrategy contract is a strategy for managing funds and interacting with the MaxApy Vault within the Unlockd Protocol.

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 maxAPY team and the maxAPY 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.

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.

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

zokyo