DetectBox
Powered by UNSNARL

#DECENTRALISED PROACTIVE AUDIT WAR-ROOMS

Decentralised Asset Management Protocol Security Assessment

KUNJI FINANCE

AUGUST 16TH, 2023

TABLE OF CONTENTS

SUMMARY

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

The auditing process pays special attention to the following considerations:

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

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

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

CERTIFICATION

This is to Certify that the **Kunji Finance Security Assessment Report** was prepared using gold standard web3 security standards with all standard and advanced checks deployed.

The **Main Auditor** Involved in the Audit was -
Samrat Gupta(@Sm4rty_)

The **Detect-Warden** Involved in the Audit was -
Rohan Jha(@rohan16_)

The Auditor mentioned hereby was chosen after self due-dilligence from the project's end.

All reported issues were either acknowledged or fixed by the project.

DetectBox's audit mitigation & residual vulnerability check did not find any major vulnerability to report within the Scope and requirements of the audit.

From
Team DetectBox

Kunji
Security Assessment Report

ABOUT DETECTBOX

DetectBox by **UNSNARL** brings to you world's first Decentralised Audit War-Rooms. An intense collaborative Environment where chosen auditors/audit team, project's decision makers, Projects developers and Detect-Wardens join hands to conduct rigorous security audits with absolute transparency followed by a **Proof of Audit (POA).**

We at DetectBox not only find bugs in smart-contracts but also take a deep look into the project's **tokenomics**, **white-paper**, **business logic**, **functional flows** and overall development health.

At DetectBox, you get to choose from a pool of Independent Security researchers which is curated by strong due-diligence. List your project requirements, verify the auditor's past work from their profiles and choose the auditor that rightly fits your project's needs and your budget.

Our auditors have successfully completed **200+ audits**, found over **2200+ vulnerabilities** and secured over **$102M+ worth of TVL.**

To know more about us visit - detectbox.io
To see our docs - https://unsnarl.gitbook.io/detectbox/

Yours Securely
UNSNARL

OVERVIEW

Audit Summary

A time-boxed independent security assessment of the Kunji Finance Contract was done by Samrat Gupta(@Sm4rty_), Rohan Jha(@rohan16___) and Team DetectBox, with a focus on the security aspects of the application's implementation.

We performed the security assessment based on the agreed scope, following our approach and methodology. Based on our scope and our performed activities, our security assessment revealed 6 Medium severity, 8 Low severity, 8 Informational and 4 Gas Optimisation security issues.

Audit Timeline: 30th July'23 - 11th August'23

Code Repository:
https://github.com/Kunji-Finance/KF-Contract

Review commit hash:
29e4fb07cb3a4e0eba477b8a7504846c2a600adf

OVERVIEW

Audit Methodology

During our security assessments, we uphold a rigorous approach to maintain high-quality standards. Our methodology encompasses thorough functional testing and meticulous manual code reviews. To ensure comprehensive issue coverage, we employ checklists derived from industry best practices and widely recognized concerns, specifically tailored to Solidity smart contract assessment.

Throughout the smart contract audit process, we prioritize the following aspects to uphold excellence:

1. Code Quality: We diligently evaluate the overall quality of the code, aiming to identify any potential vulnerabilities or weaknesses.
2. Best Practices: Our assessments emphasize adherence to established best practices, ensuring that the smart contract follows industry-accepted guidelines and standards.
3. Documentation and Comments: We meticulously review code documentation and comments to ensure they accurately reflect the underlying logic and expected behaviour of the contract.

Auditing smart contracts involves a comprehensive analysis of the code to identify potential vulnerabilities and security risks. To achieve comprehensive coverage, we employ a series of security checklist tables, each addressing specific areas of concern. These include:

- System / Platform
- Access Control
- Storage

OVERVIEW

- Gas Issues and Efficiency
- Code Issues
- Error Handling and Exception Handling:
- Transaction Handling
- Entrypoint Validation
- Administration and Operator Functions
- Additional Topics and Test Cases

Vulnerability Summary

Severity classification

Severity	Impact : High	Impact : Medium	Impact : Low
Likelihood : High	Critical	High	Medium
Likelihood : Medium	High	Medium	Low
Likelihood : Low	Medium	Low	Low

Findings Summary

High	0 issues
Medium	6 issues
Low	8 issues
Informational	8 issues
Gas Optimisations	4 issues

OVERVIEW

Audit Scope

The code under review is composed of multiple smart contracts written in the Solidity language and includes 3580 nLOC-normalized source lines of code (only source-code lines).

File	nLines	nSLOC	Complex. Score
contracts/UsersVault.sol	709	557	256
contracts/ContractsFactory.sol	305	223	247
contracts/adapters/Lens.sol	607	490	298
contracts/adapters/gmx/GMXAdapter.sol	890	668	268
contracts/adapters/gmx/interfaces/IGmxPositionManager.sol	6	3	9
contracts/adapters/gmx/interfaces/IGmxAdapter.sol	20	15	21
contracts/adapters/gmx/interfaces/IGmxReader.sol	6	3	9
contracts/adapters/gmx/interfaces/IGmxRouter.sol	40	33	38
contracts/adapters/gmx/interfaces/IVaultPriceFeed.sol	6	3	33
contracts/adapters/gmx/interfaces/IGmxOrderBook.sol	32	27	49
contracts/adapters/gmx/interfaces/IGmxVault.sol	6	3	21
contracts/Observers/GMXObserver.sol	149	129	87
contracts/adapters/uniswap/libraries/BytesLib.sol	112	65	141
contracts/adapters/uniswap/UniswapV3Adapter.sol	287	234	101
contracts/adapters/uniswap/interfaces/IQuoterV2.sol	51	45	9
contracts/adapters/uniswap/interfaces/IUniswapV3Router.sol	57	49	23
contracts/adapters/uniswap/interfaces/IUniswapV3Factory.sol	35	27	13
contracts/adapters/uniswap/interfaces/INonfungiblePositionManager.sol	131	119	28

OVERVIEW

Audit Scope

contracts/adapters/uniswap/interfaces/IUniswapV3Adapter.sol	7	3	9
contracts/adapters/uniswap/interfaces/IUniswapV3Pool.sol	11	6	37
contracts/TraderWallet.sol	507	374	252
contracts/DynamicValuation.sol	193	148	94
contracts/BaseVault.sol	180	133	80
contracts/interfaces/IBaseVault.sol	6	3	15
contracts/interfaces/IDynamicValuation.sol	29	19	21
contracts/interfaces/ITraderWallet.sol	9	5	63
contracts/interfaces/IObserver.sol	6	3	5
contracts/interfaces/IAdaptersRegistry.sol	6	3	5
contracts/interfaces/IAdapter.sol	17	7	3
contracts/interfaces/IContractsFactory.sol	54	49	77
contracts/interfaces/ILens.sol	21	17	3
contracts/interfaces/Errors.sol	37	33	1
contracts/interfaces/IUsersVault.sol	19	13	39
contracts/interfaces/IPlatformAdapter.sol	14	9	5
contracts/interfaces/Events.sol	67	62	1
Totals	**4632**	**3580**	**2361**

Auditors Involved :

Main Auditor -

Samrat Gupta - https://app.detectbox.io/profile/sm4rty

Detect Warden -

Rohan Jha - https://app.detectbox.io/profile/Rohan16

FINDINGS

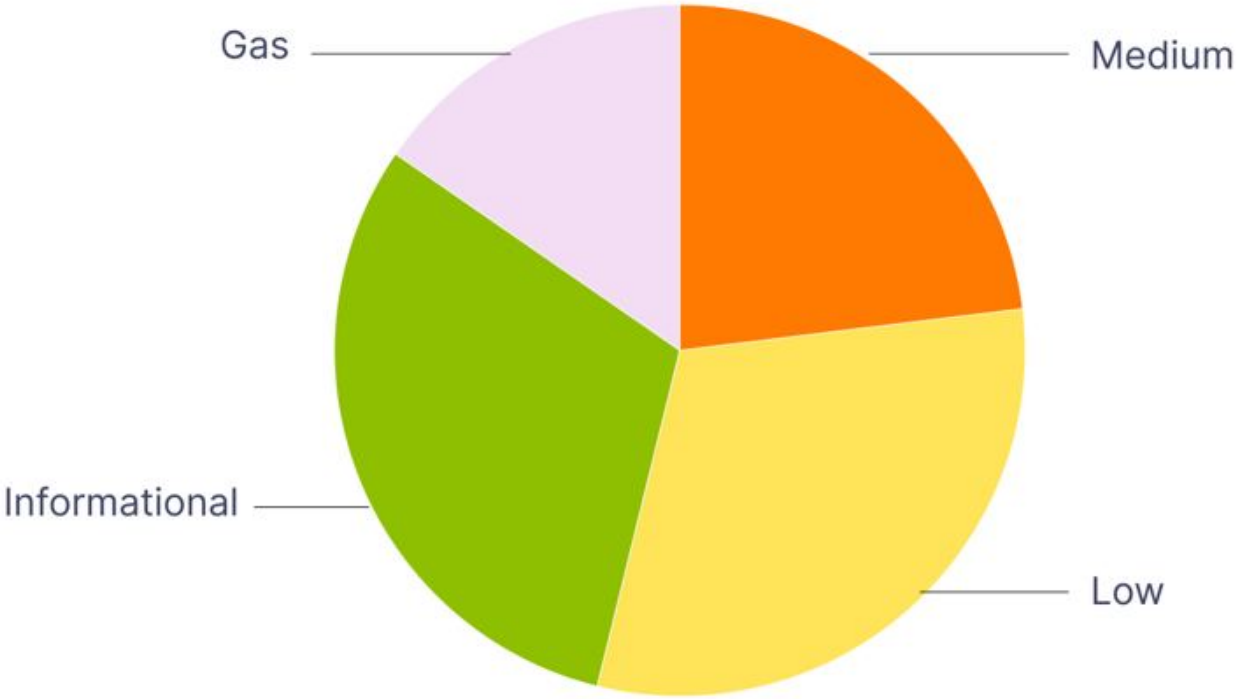
Detailed Summary of Findings

Sl. No.	Name	Severity
M-01	Unsafe ERC20 Transfer Function Usage	Medium
M-02	Chainlink Sequencer Uptime Check Not Implemented Properly	Medium
M-03	Use call instead of Transfer for address payable	Medium
M-04	The owner is a single point of failure and a centralization risk	Medium
M-05	Non-Upgradable OpenZeppelin Contracts Used in Upgradeable Contracts	Medium
M-06	Missing Access control in emergencyClose() function	Medium
L-01	No logic is implemented to handle profit in rollover function.	Low
L-02	Natspec is missing	Low
L-03	Missing Zero-Address Check in setGmxObserver() Function	Low
L-04	No error or revert in claim function if claim ==0	Low
L-05	Avoid the use of Floating Pragma	Low
L-06	Consider implementing two-step procedure for updating protocol addresses	Low
L-07	Unused receive() function will lock Ether in contract	Low
L-08	Use Ownable2StepUpgradeable rather than OwnableUpgradeable	Low

FINDINGS

Detailed Summary of Findings

Sl. No.	Name	Severity
I-01	Removal of Commented-Out Code for Better Code Quality	Informational
I-02	Open TODOs	Informational
I-03	Variable names don't follow the Solidity style guide	Informational
I-04	Using unnamed mappings	Informational
I-05	Consider adding emergency-stop functionality	Informational
I-06	Imports could be organized more systematically	Informational
I-07	Event is not properly indexed	Informational
I-08	Typos	Informational
G-01	Explicitly initializing variables with their default values wastes gas	Gas
G-02	Use assembly to check for address(0)	Gas
G-03	Caching the array length outside a loop	Gas
G-04	Redundant zero-address check in the removeTrader() function	Gas



FINDINGS

Static Analysis

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

FINDINGS

Manual Review

Medium Severity Issues

M-01. Unsafe ERC20 Transfer Function Usage

Description :

It is not safe to use the ERC20 transfer function without checking the results. In BaseVault.sol, emergencyWithdraw() function uses the transfer() method to transfer funds from users to the contract. However, due to some ERC20 tokens' non—compliance with the standards, if a non standard token like is used, the input will return nothing instead of a boolean value. As a result, the condition checking for if(!success) where is the return value of will always be triggered and revert the transaction.

Code Snippets :

```
function emergencyWithdraw(IERC20 token) external onlyOwner {
        if (address(token) == 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE) {
            uint256 ethBalance = address(this).balance;
            payable(owner()).transfer(ethBalance);
            return;
        }
        uint256 balance = token.balanceOf(address(this));
        token.transfer(owner(), balance); //@audit: Use safeTransfer instead of tr
ansfer
    }
```

FINDINGS

```
function collectFees(uint256 amount) external override onlyOwner {
        uint256 _kunjiFeesAssets = kunjiFeesAssets;

        if (amount > _kunjiFeesAssets) {
            revert TooBigAmount();
        }

        kunjiFeesAssets = _kunjiFeesAssets - amount;

        address feeReceiver = IContractsFactory(contractsFactoryAddress)
            .feeReceiver();
        IERC20(underlyingTokenAddress).transfer(feeReceiver, amount);  //@audit: U
se safeTransfer instead of transfer
    }
```

Recommendations :

I recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

M-02.Chainlink Sequencer Uptime Check Not Implemented Properly

FINDINGS

Description :

In the DynamicValuation contract is that the *_checkSequencerUptimeFeed()* function is only implemented in the *_getUSDValueOfAddress()* function. This means that the sequencer uptime is not checked when fetching price feeds in the *_getDataFeedAnswer()* function. *_getDataFeedAnswer* function is called in *getOraclePrice()* to fetch price in different parts of contract. It leads to insufficient checks for if the sequencer is active. If the sequencer is down, then the price feeds returned by the *_getDataFeedAnswer()* function will be incorrect. This could lead to incorrect valuations of tokens and assets, which could have financial consequences for users of the DynamicValuation contract.

Code Snippets :

```
function _getDataFeedAnswer(
        OracleData memory oracleData,
        address token
    ) private view returns (uint256) {
        if (oracleData.dataFeed == address(0)) {
            revert NoOracleForToken(token);
        }

        AggregatorV2V3Interface _dataFeed = AggregatorV2V3Interface(
            oracleData.dataFeed
        );

        (, int answer, , uint256 updatedAt, ) = _dataFeed.latestRoundData(); //24025762183
        if (answer <= 0) {
            revert BadPrice();
        }
        if (block.timestamp - updatedAt > oracleData.heartbeat) {
            revert TooOldPrice();
        }

        return uint256(answer);
    }
```

FINDINGS

Recommendation :

Add a line to check sequencer uptime in _getDataFeedAnswer() function.

```
function _getDataFeedAnswer(
    OracleData memory oracleData,
    address token
) private view returns (uint256) {
    if (oracleData.dataFeed == address(0)) {
        revert NoOracleForToken(token);
    }

    _checkSequencerUptimeFeed(); // Add this line to check sequencer uptime

    AggregatorV2V3Interface _dataFeed = AggregatorV2V3Interface(
        oracleData.dataFeed
    );

    (, int answer, , uint256 updatedAt, ) = _dataFeed.latestRoundData(); //24025762183
    if (answer <= 0) {
        revert BadPrice();
    }
    if (block.timestamp - updatedAt > oracleData.heartbeat) {
        revert TooOldPrice();
    }

    return uint256(answer);
}
```

M-03. Use call instead of Transfer for address payable

FINDINGS

Description :

The *emergencyWithdraw()* function in the BaseVault.sol contract uses the transfer() function to send ether to an address payable. The transfer() function forwards a fixed amount of 2300 gas. This means that the function may fail if the gas cost of EVM instructions changes significantly during a hard fork, or if the claimer smart contract does not implement a payable function or does implement a payable fallback function that uses more than 2300 gas units.

Code Snippets :

```
function emergencyWithdraw(IERC20 token) external onlyOwner {
        if (address(token) == 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE) {
            uint256 ethBalance = address(this).balance;
            payable(owner()).transfer(ethBalance); // @audit-issue  Use call instead of transf
er for address payable
            return;
        }
        uint256 balance = token.balanceOf(address(this));
        token.transfer(owner(), balance);
    }
```

Recommendations :

Instead use the .call() function to transfer ether and avoid some of the limitations of .transfer().

Gas units can also be passed to the .call() function as a variable to accommodate any uses edge cases. Gas could be a mutable state variable that can be set by the contract owner.

FINDINGS

```
(bool success, ) = payable(owner()).call{value: ethBalance}("");
require(success, "Transfer failed.");
```

M-04. The owner is a single point of failure and a centralization risk

Description :

The owner of the BaseVault.sol contract has the ability to withdraw all funds from the contract. This creates a single point of failure and centralization risk. If the owner is malicious, they could steal all funds from the contract.

Code Snippets :

```
function emergencyWithdraw(IERC20 token) external onlyOwner { //Add a timelock function for i
t.
        if (address(token) == 0xEeeeeEeeeEeEeeEeEeEeeeEEEeeeeEeeeeeeeEEeE) {
            uint256 ethBalance = address(this).balance;
            payable(owner()).transfer(ethBalance);
            return;
        }
        uint256 balance = token.balanceOf(address(this));
        token.transfer(owner(), balance);
    }
```

FINDINGS

Recommendation :

A timelock should be implemented for the *emergencyWithdraw()* function. This would prevent the owner from withdrawing funds immediately.

M-05. Non-Upgradable OpenZeppelin Contracts Used in Upgradeable Contracts

Description :

The contracts ContractFactory.sol and TraderWallet.sol utilize an upgradable version of OpenZeppelin contracts such as OwnableUpgradeable and ReentrancyGuardUpgradeable, which ensures that the code is safe for upgradeability. However, these contracts import the non-upgradable versions of SafeERC20, IERC20Metadata and EnumerableSet from the OpenZeppelin library which is not a good practice.
An upgradable version of the contract ensures that the code is safe for upgradeability. Check out here for more details.

Code Snippets :

```
7: import {EnumerableSet} from "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";
8  import {IERC20Metadata} from "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metada
ta.sol";
```

FINDINGS

```
6  import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
7: import {EnumerableSet} from "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";
```

Recommendation :

To ensure full compatibility with the upgradeable design pattern and to enhance the contract's upgradeability, it is recommended to use the upgradable versions of contracts consistently throughout the codebase.

```
import {SafeERC20Upgradeable} from "@openzeppelin/contracts-upgradeable/token/ERC20/utils/Safe
ERC20Upgradeable.sol";
import {EnumerableSetUpgradeable} from "@openzeppelin/contracts-upgradeable/utils/structs/Enum
erableSetUpgradeable.sol";
```

M-06. Missing Access control in emergencyClose() function.

Description :

The *emergencyClose()* function in the UserVault.sol contract lacks proper access control. The function visibility is marked as external, which means that any external user can call this function. Additionally, there are no validation checks in the function to verify that the caller is authorized to close the user's position.

FINDINGS

Code Snippets :

```
function emergencyClose() external { //@access control issue?
        address _traderWalletAddress = traderWalletAddress;
        if (
            ITraderWallet(_traderWalletAddress).lastRolloverTimestamp() +
                emergencyPeriod >
            block.timestamp &&
            !isEmergencyOpen
        ) revert TooEarly();
        bool isRequestFulfilled = _closeUniswapPositions(_traderWalletAddress);
        _closeGmxPositions(_traderWalletAddress);

        if (!isRequestFulfilled) {
            currentSlippage += slippageStepPercent;
            isEmergencyOpen = true;
        } else {
            currentSlippage = defaultSlippagePercent;
            isEmergencyOpen = false;
        }
    }
```

Recommendation :

Implement proper access control so that no unauthorized caller can access emergency functions. Additionally, The function should be modified to include validation checks to verify that the caller is authorized to close the user's position.

Low Severity Issues

L-01. No logic is implemented to handle profit in rollover function

FINDINGS

Description :

The rollover() function in Traderwallet.sol calculates the profit and has an if check for overallProfit > 0. However, the function does not implement any logic to handle the profit, and nothing is executed inside the if statement. There is a comment to do something with the profit, but no implementation is present inside the code block.

Code Snippets :

```
// not sure if the execution is here. Don't think so
function rollover() external override {
    if (lastRolloverTimestamp + rolloverPeriod > block.timestamp) {
        revert TooEarly();
    }

    uint256 _cumulativePendingDeposits = cumulativePendingDeposits;
    uint256 _cumulativePendingWithdrawals = cumulativePendingWithdrawals;

    ...
    ...

    // get profits
    int256 overallProfit;
    if (_currentRound != 0) {
        overallProfit =
            int256(_newAfterRoundBalance) -
            int256(afterRoundBalance); // 0 <= old < new => overallProfit = new - old > 0
    }
    if (overallProfit > 0) { @audit Profit calculated but no logic is implemented
        // DO SOMETHING HERE WITH PROFIT
        // PROFIT IS CALCULATED IN ONE TOKEN
        // BUT PROFIT IS DISTRIBUTED AMONG OPEN POSITIONS
        // AND DIFFERENT TOKEN BALANCES
    }
```

Recommendation :

The function does not have any fee handling logic based on profits. Therefore, we recommend to add logic accordingly or to remove the if statement as if it does not have any purpose.

FINDINGS

L-02. Missing Natspec in many function

Description :

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).
There are multiple functions in few smart contracts in scope that lack NatSpec comments, which are essential for providing clear and comprehensive documentation for functions, return variables, and other contract elements.

Code Snippets :

The following smart contracts in scope lack NatSpec comments in multiple functions:
https://github.com/Kunji-Finance/KF-Contract/blob/develop/contracts/ContractsFactory.sol
https://github.com/Kunji-Finance/KF-Contract/blob/develop/contracts/DynamicValuation.sol
https://github.com/Kunji-Finance/KF-Contract/blob/develop/contracts/TraderWallet.sol
https://github.com/Kunji-Finance/KF-Contract/blob/develop/contracts/UsersVault.sol

Recommendation :

To enhance the quality and usability of the smart contracts, it is strongly recommended to add NatSpec comments to various parts of the code.

FINDINGS

L-03. Missing Zero-Address Check in setGmxObserver() Function

Description

The setGmxObserver() function in the DynamicValuation.sol contract lacks a check for a zero address (0×0) when setting the gmxObserver value. However, many other setter functions in the contract properly include a check for a zero address.

Code Snippets :

```
function setGmxObserver(address newValue) external override onlyOwner {
        gmxObserver = newValue; //@audit no check for 0-address.

        emit SetGmxObserver(newValue);
    }
```

Recommendation :

The recommended mitigation steps are to add a check for 0-addresses to the setGmxObserver() function.

```
function setGmxObserver(address newValue) external override onlyOwner {
    _checkZeroAddress(newValue, "_GmxObserver");
     gmxObserver = newValue;

     emit SetGmxObserver(newValue);
    }
```

FINDINGS

L-04. No error or revert in claim function if claim ==0.

Description :

The claim() function in UserVault.sol does not check if the unclaimedDepositShares or unclaimedWithdrawAssets variables are equal to 0. If these variables are equal to 0, the function will silently execute without any error or revert.

Code Snippets :

```
function claim() external override {
    UserData memory data = _updateUserData(msg.sender);

    if (data.unclaimedDepositShares > 0) {
        super._transfer(
            address(this),
            msg.sender,
            data.unclaimedDepositShares
        );

        delete _userData[msg.sender].unclaimedDepositShares;

        emit SharesClaimed(
            data.round,
            data.unclaimedDepositShares,
            msg.sender,
            msg.sender
        );
    }

    if (data.unclaimedWithdrawAssets > 0) {
        uint256 underlyingBalance = IERC20(underlyingTokenAddress)
            .balanceOf(address(this));

        // LOGIC HERE

        emit AssetsClaimed(
            data.round,
            transferAmount,
            msg.sender,
            msg.sender
        );
    }
}
```

FINDINGS

Recommendation :

To mitigate this vulnerability, the claim() function should be updated to check if the unclaimedDepositShares and unclaimedWithdrawAssets variables are equal to 0. If they are equal to 0, the function should revert with an error.

L-05. Floating Pragma

Description :

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma.

Code Snippets :

```
contracts/adapters/gmx/interfaces/IGmxOrderBook.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/gmx/interfaces/IGmxPositionManager.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/gmx/interfaces/IGmxReader.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/gmx/interfaces/IGmxRouter.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/gmx/interfaces/IGmxVault.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/gmx/interfaces/IVaultPriceFeed.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/uniswap/interfaces/INonfungiblePositionManager.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/uniswap/interfaces/IQuoterV2.sol:
  3: pragma solidity >=0.8.0;
```

FINDINGS

```
contracts/adapters/uniswap/interfaces/IUniswapV3Factory.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/uniswap/interfaces/IUniswapV3Pool.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/uniswap/interfaces/IUniswapV3Router.sol:
  3: pragma solidity >=0.8.0;

contracts/adapters/uniswap/libraries/BytesLib.sol:
  10: pragma solidity >=0.8.0 <0.9.0;

contracts/interfaces/IAdapter.sol:
  3: pragma solidity >=0.8.0;
```

```
contracts/interfaces/IAdaptersRegistry.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/IBaseVault.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/IContractsFactory.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/IDynamicValuation.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/ILens.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/IObserver.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/IPlatformAdapter.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/ITraderWallet.sol:
  3: pragma solidity >=0.8.0;

contracts/interfaces/IUsersVault.sol:
  3: pragma solidity >=0.8.0;
```

Recommendation :

Consider locking the pragma in all the contract.

FINDINGS

L-06. Consider implementing two-step procedure for updating protocol addresses

Description :

Certain functions, such as setGmxObserver and setFeeReceiver, allow an address to be set in a single step. This approach can be error-prone. The critical procedures should be a two-step process.

Code Snippets :

```
 function setGmxObserver(address newValue) external override onlyOwner
 {
         gmxObserver = newValue;

         emit SetGmxObserver(newValue);
     }

function setFeeReceiver(
        address newFeeReceiver
    ) external override onlyOwner {
        _checkZeroAddress(newFeeReceiver, "newFeeReceiver");

        feeReceiver = newFeeReceiver;

        emit FeeReceiverSet(newFeeReceiver);
    }
```

Recommendation :

The lack of a two-step procedure for critical operations leaves them error-prone. Consider adding two-step procedure on the critical functions.

FINDINGS

L-07. Unused receive() function will lock Ether in contract

Description :

The BaseVault.sol contract contains an unused receive() function. This function is payable, meaning that it can receive Ether. However, the function does not do anything with the Ether that it receives. This means that if an user sends Ether to the contract using the receive() function, the Ether will be locked in the contract and cannot be recovered.

Code Snippets :

```
receive() external payable {}
```

Recommendation :

The following mitigation steps can be taken to address this vulnerability:

1. Remove the receive() function from the contract.
2. If the receive() function is needed, then modify the function to do something with the Ether that it receives.

FINDINGS

L-08. Use Ownable2StepUpgradeable rather than OwnableUpgradeable

Description :

The BaseVault.sol, DynamicValuation.sol, and ContractsFactory.sol contracts all use the Ownable contract from OpenZeppelin. However, the Ownable contract does not prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address). This could result in the contract being taken over by an attacker.

Code Snippets :

```
BaseVault.sol:5:import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.so
l";
DynamicValuation.sol:7:import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeab
le.sol";
ContractsFactory.sol:5:import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeab
le.sol";
```

Recommendation :

Use the Ownable2Step or Ownable2StepUpgradeable contract instead of the Ownable contract. These contracts prevent the contract ownership from being transferred to an address that cannot handle it

FINDINGS

I-01. Removal of Commented-Out Code for Better Code Quality

Description :

The contracts TraderWallet.sol and Lens.sol contains lines of code that are commented out but serve no purpose. Commented-out code segments do not contribute to the functionality of the contracts but can clutter the codebase, making it harder for developers to understand the actual logic of the contracts. These lines of code should be removed to improve the overall code quality and clarity of the contracts.

Code Snippets :

```
//     address tokenIn,
//     address tokenOut
// ) external view returns (uint256 amountIn) {
//     return IGmxReader(gmxReader).getMaxAmountIn(address(gmxVault), tokenIn, tokenOut);
// }

// /// @notice Returns amount out after fees and the fee amount
// /// @param tokenIn The address of input token
// /// @param tokenOut The address of output token
// /// @param amountIn The amount of tokenIn to be swapped
// /// @return amountOutAfterFees The amount out after fees,
// /// @return feeAmount The fee amount in terms of tokenOut
// function getAmountOut(
//     address tokenIn,
//     address tokenOut,
//     uint256 amountIn
// ) external view returns (uint256 amountOutAfterFees, uint256 feeAmount) {
//     return
//         IGmxReader(gmxReader).getAmountOut(
//             address(gmxVault),
```

Recommendation :

To enhance the code quality and maintainability of the contracts, it is recommended to remove the commented-out code segment

FINDINGS

I-02. Open TODOs

Description :

The BaseVault, IGmxAdapter and few other contracts contain TODO comments. These comments indicate that there are open questions or issues that need to be resolved before the contracts can be deployed.

Code Snippets :

```
contracts/BaseVault.sol:
  169
  170:     // @todo consider adding TimeLock for this function?
  171      /// @notice Withdrawing tokens in the emergency case from the contract

contracts/adapters/gmx/interfaces/IGmxAdapter.sol:
  62      /// @notice Returns data for open position
  63:     // todo
  64      function getPosition(uint256) external view returns (uint256[] memory);

tests/adapters/uniswapV3-adapter.ts:
  959
  960: // @todo add tests with ratio < 1e18
  961
```

Recommendation :

Remove the TODO comments or resolve them.

FINDINGS

I-03. Variable names don't follow the Solidity style guide

Description :

The Solidity style guide specifies that constant variable names should be in CONSTANT_CASE, which means that each word should be capitalized and underscores should be used to separate the words. However, there are multiple instances in the contract where this pattern is not followed.

Code Snippets :

```
DynamicValuation.sol:28:    uint8 public constant override decimals = 30;
Observers/GMXObserver.sol:14:    IGmxVault public constant gmxVault =
Observers/GMXObserver.sol:16:    IGmxReader public constant gmxReader =
Observers/GMXObserver.sol:18:    IGmxOrderBook public constant gmxOrderBook =
Observers/GMXObserver.sol:22:    uint8 public constant override decimals = 30;
adapters/uniswap/UniswapV3Adapter.sol:35:    IUniswapV3Router public constant uniswapV3Router =
adapters/uniswap/UniswapV3Adapter.sol:37:    IQuoterV2 public constant quoter =
adapters/uniswap/UniswapV3Adapter.sol:40:    uint256 public constant ratioDenominator = 1e18;
adapters/uniswap/UniswapV3Adapter.sol:43:    uint128 public constant slippageAllowanceMax = 3e17;
// 30%
adapters/uniswap/UniswapV3Adapter.sol:46:    uint128 public constant slippageAllowanceMin = 1e15;
// 0.1%
adapters/gmx/GMXAdapter.sol:21:    address internal constant gmxRouter =
adapters/gmx/GMXAdapter.sol:23:    address internal constant gmxPositionRouter =
adapters/gmx/GMXAdapter.sol:25:    IGmxVault internal constant gmxVault =
adapters/gmx/GMXAdapter.sol:27:    address internal constant gmxOrderBook =
adapters/gmx/GMXAdapter.sol:29:    address internal constant gmxOrderBookReader =
adapters/gmx/GMXAdapter.sol:31:    address internal constant gmxReader =
adapters/gmx/GMXAdapter.sol:35:    uint256 private constant ratioDenominator = 1e18;
adapters/gmx/GMXAdapter.sol:38:    uint256 public constant slippage = 1e17; // 10%
```

Recommendation :

Rename the variables to follow the Solidity style guide.

FINDINGS

I-04. Using unnamed mappings

Description :

The contracts use unnamed mappings, which can make it difficult to understand the purpose of each mapping. Consider using named mappings to make it easier to understand the purpose of each mapping. This can make the code less readable and maintainable.

Code Snippets :

```
DynamicValuation.sol:35:    mapping(address => OracleData) private _chainlinkOracles;
TraderWallet.sol:39:    mapping(address => mapping(address => bool)) public override gmxShortPair
s;
ContractsFactory.sol:28:    mapping(address => bool) public override isTraderWallet;
ContractsFactory.sol:31:    mapping(address => bool) public override isUsersVault;
ContractsFactory.sol:33:    mapping(address => bool) public override allowedInvestors;
ContractsFactory.sol:34:    mapping(address => bool) public override allowedTraders;
UsersVault.sol:47:    mapping(uint256 => uint256) public assetsPerShareXRound;
UsersVault.sol:49:    mapping(address => UserData) private _userData;
UsersVault.sol:50:    mapping(uint256 => uint256) private _underlyingPriceXRound;
```

Recommendation :

Rename the mappings to make it clear what the purpose of each mapping is.

FINDINGS

I-05. Consider adding emergency-stop functionality

Description :

The contracts do not have a global emergency halt mechanism. This means that if there is a hack or other emergency, the only way to stop the protocol is to pause individual contracts one-by-one. This can be time-consuming and stressful, and it may not be possible to pause all of the contracts before the hack is complete.
Similar Issue can be found here.

Recommendation :

- Add a global emergency halt mechanism to the contracts.
- Document the emergency halt mechanism so that users know how to use it.

I-06. Imports could be organized more systematically

Description :

The imports in the contracts are not organized in a systematic way. The contract's interface should be imported first, followed by each of the interfaces it uses, followed by all other files. However, the imports in the contract are not in this order.
Similar findings in a Code4rena Contest.

FINDINGS

Code Snippets :

```
import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.so
l";
import {ReentrancyGuardUpgradeable} from "@openzeppelin/contracts-upgradeable/security/ReentrancyG
uardUpgradeable.sol";

import {IERC20Metadata} from "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";

import {GMXAdapter} from "./adapters/gmx/GMXAdapter.sol";

import {Events} from "./interfaces/Events.sol";
import {Errors} from "./interfaces/Errors.sol";

import {IAdaptersRegistry} from "./interfaces/IAdaptersRegistry.sol";
import {IContractsFactory} from "./interfaces/IContractsFactory.sol";
import {IDynamicValuation} from "./interfaces/IDynamicValuation.sol";
import {IAdapter} from "./interfaces/IAdapter.sol";
import {IBaseVault} from "./interfaces/IBaseVault.sol";
import {IGmxVault} from "./adapters/gmx/interfaces/IGmxVault.sol";
import {IERC20} from "@openzeppelin/contracts/interfaces/IERC20.sol";
```

Recommendation :

- Use namespaces, import groups, and aliases to help organize the imports.
- Organize the imports in a systematic way.

I-07. Event is not properly indexed

Description :

Multiple events in the Events.sol contract are not indexed efficiently. Each event only indexes one field, when it could index three fields. This means that off-chain tools that parse events will have to scan more data to find the information they need.

FINDINGS

Code Snippets :

```
event SharesClaimed(
        uint256 round,
        uint256 shares,
        address caller,
        address receiver
    );

    event AssetsClaimed(
        uint256 round,
        uint256 assets,
        address owner,
        address receiver
    );

    event UsersVaultRolloverExecuted(
        uint256 round,
        uint256 underlyingTokenPerShare,
        uint256 sharesToMint,
        uint256 sharesToBurn,
        int256 overallProfit,
        uint256 unusedFunds
    );

 event OperationExecuted(
         uint256 protocolId,
         uint256 timestamp,
         string target,
         bool replicate,
         uint256 walletRatio
     );

     event TraderWalletRolloverExecuted(
         uint256 timestamp,
         uint256 round,
         int256 traderProfit,
         uint256 unusedFunds
     );
```

Recommendation :

It can be mitigated by adding indexing to the remaining fields in the above events.

FINDINGS

I-08. Typos

Description :

The comment on line 10 of the IAdapter.sol contract contains a typo. The word "signature" is misspelled as "signatura".

Code Snippets :

```
// signatura of the funcion
```

Gas Optimizations

G-01. Explicitly initializing variables with their default values wastes gas.

Description :

The DynamicValuation.sol, GMXAdapter.sol, Lens.sol, ContractsFactory.sol, and UsersVault.sol contracts all contain variables that are initialized with their default values. For example, the UsersVault.sol contract contains collateralDelta variable, which is initialized to 0.
The default value of a uint256 variable is 0, so explicitly initializing collateralDelta with 0 is unnecessary. This wastes gas, as the Solidity compiler has to store the value 0 in memory even though it is not actually used.

FINDINGS

Code Snippets :

```
DynamicValuation.sol:138:        for (uint256 i = 0; i < tokens.length; ++i) {
adapters/gmx/GMXAdapter.sol:83:        if (uint256(traderOperation.operationId) == 0) {
adapters/Lens.sol:236:        for (uint256 i = 0; i < lengthShorts; ++i) {
adapters/Lens.sol:259:        for (uint256 i = 0; i < totalLength; ++i) {
ContractsFactory.sol:75:        for (uint256 i = 0; i < length; ++i) {
ContractsFactory.sol:108:        for (uint256 i = 0; i < length; ++i) {
UsersVault.sol:753:        for (uint256 i = 0; i < positions.length; ++i) {
UsersVault.sol:756:            uint256 collateralDelta = 0;
```

Recommendation :

Declare variables without initializing them. We can use uint number; instead of uint number = 0;

G-02. Use assembly to check for address(0).

Description :

The BaseVault.sol contract contains the _checkZeroAddress() function, which checks if the supplied address is not a 0-address. However, the current implementation of the function is not the most gas efficient. By using assembly, the function can be made more gas efficient.

Code Snippets :

```
function _checkZeroAddress(
        address _variable,
        string memory _message
    ) internal pure {
        if (_variable == address(0)) revert ZeroAddress({target: _message});
    }
```


FINDINGS

Recommendation :

The following code snippet shows how to use assembly to check for a 0-address:

```
assembly {
if iszero(_addr) {
mstore (0x00, "zero address")
revert (0x00, 0x20)
}
```

G-03. Caching the array length outside a loop

Description :

The DynamicValuation.sol, TraderWallet.sol, Observers/GMXObserver.sol, adapters/gmx/GMXAdapter.sol, adapters/Lens.sol, ContractsFactory.sol, and UsersVault.sol contracts all contain loops that iterate over arrays. In each loop, the length of the array is read. However, the length of the array does not change during the loop, so it can be cached outside the loop. This would save 3 gas per iteration, or a total of 27 gas per loop.

FINDINGS

Code Snippets :

```
DynamicValuation.sol:138:        for (uint256 i = 0; i < tokens.length; ++i) {
TraderWallet.sol:130:        for (uint256 i; i < collateralTokens.length; ) {
TraderWallet.sol:163:        for (uint256 i; i < _tokens.length; ) {
Observers/GMXObserver.sol:91:        for (uint256 i; i < isLong.length; ) {
adapters/gmx/GMXAdapter.sol:815:            for (uint256 i; i < allowedTradeTokens.length; ) {
adapters/Lens.sol:236:        for (uint256 i = 0; i < lengthShorts; ++i) {
adapters/Lens.sol:243:        for (uint256 i = lengthShorts; i < totalLength; ++i) {
adapters/Lens.sol:259:        for (uint256 i = 0; i < totalLength; ++i) {
ContractsFactory.sol:75:          for (uint256 i = 0; i < length; ++i) {
ContractsFactory.sol:108:         for (uint256 i = 0; i < length; ++i) {
ContractsFactory.sol:199:         for (uint256 i; i < _tokens.length; ) {
UsersVault.sol:672:        for (uint256 i = 1; i < tokens.length; ++i) {
UsersVault.sol:753:        for (uint256 i = 0; i < positions.length; ++i) {
```

Recommendation :

The following code snippet shows how to cache the array length outside a loop:

```
uint256 length = tokens.length;
for (uint256 i = 0; i < length; ++i) {
  // do something with the token at index i
}
```

FINDINGS

G-04. Redundant zero-address check in the removeTrader() function

Description :

The removeTrader() function in the UsersVault.sol contract contains a zero-address check. This check ensures that the address passed to the function is not a zero address. However, this check is redundant, as the addTrader() function already performs a zero-address check before adding an address to the allowedTraders array.

Code Snippets :

https://github.com/Kunji-Finance/KF-Contract/blob/29e4fb07cb3a4e0eba477b8a7504846c2a600adf/contracts/ContractsFactory.sol#L124

Recommendation :

Remove the zero-address check from the removeTrader() function.

POST AUDIT CONCLUSION

Fixing the Findings

Sl. No.	Name	Status
M-01	Chainlink Sequencer Uptime Check Not Implemented Properly	Fixed
M-02	Unsafe ERC20 Transfer Function Usage	Fixed
M-03	Use call instead of Transfer for address payable	Fixed
M-04	The owner is a single point of failure and a centralization risk	Acknowledged
M-05	Non-Upgradable OpenZeppelin Contracts Used in Upgradeable Contracts	Fixed
M-06	Missing Access control in emergencyClose() function	Acknowledged
L-01	No logic is implemented to handle profit in rollover function	Acknowledged
L-02	Natspec is missing	Acknowledged
L-03	Missing Zero-Address Check in setGmxObserver() Function	Fixed
L-04	No error or revert in claim function if claim ==0	Fixed
L-05	Avoid the use of Floating Pragma	Acknowledged
L-06	Consider implementing two-step procedure for updating protocol addresses	Acknowledged
L-07	Unused receive() function will lock Ether in contract	Acknowledged
L-08	Use Ownable2StepUpgradeable rather than OwnableUpgradeable	Fixed

POST AUDIT CONCLUSION

Fixing the Findings

Sl. No.	Name	Status
I-01	Removal of Commented-Out Code for Better Code Quality	Acknowledged
I-02	Open TODOs	Acknowledged
I-03	Variable names don't follow the Solidity style guide	Acknowledged
I-04	Using unnamed mappings	Acknowledged
I-05	Consider adding emergency-stop functionality	Acknowledged
I-06	Imports could be organized more systematically	Acknowledged
I-07	Event is not properly indexed	Acknowledged
I-08	Typos	Fixed
G-01	Explicitly initializing variables with their default values wastes gas	Fixed
G-02	Use assembly to check for address(0)	Acknowledged
G-03	Caching the array length outside a loop	Fixed
G-04	Redundant zero-address check in the removeTrader() function	Fixed

APPENDIX

Finding Categories

Finding Categories

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

Gas Optimization

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

Logical Issue

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

Language Specific

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

Coding Style

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

DISCLAIMER

DetectBox (by UNSNARL) has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. UNSNARL and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

DISCLAIMER

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. UNSNARL is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. UNSNARL's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

UNSNARL in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will UNSNARL, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

DISCLAIMER

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

UNSNARL will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

UNSNARL will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.

GLORY OF AUDITORS AT DETECTBOX

DetectBox has created a pool of best auditors across the globe with significant experience in the web3 security industry auditing multiple protocols across multiple chains. Making audits better through **Proof of Talent**.

217+

Projects Audited

$100M+

Amount Secured

2200+

Vulnerabilities Detected

Follow Our Journey on

@unsnarl_secure

UNSNARL

founders@unsnarl.io

www.detectbox.io