HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Yieldster
Date: May 26th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Yieldster.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token; AMM, Liquidity Pool
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.yieldster.io
Timeline	22.04.2022 - 26.05.2022
Changelog	06.05.2022 - Initial Review 26.05.2022 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Yieldster (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope
Repository:
https://github.com/Yieldster-Framework/yieldster-contracts
Commit:
0f4b22a35beeee17d3890586f0c8ea883a9517c0
Technical Documentation: Yes - https://docs.google.com/document/d/1b4riPQvwaibi1P4IdLNnL13EZYV8wxJ2/
JS tests: Yes
Contracts:
File: ./vault/contracts/common/MasterCopy.sol
SHA3: 5d34992d913eb957d82965a9b40affe8c3c52f44d41649ab32d12908fbd9b26d

File: ./vault/contracts/exchange/Exchange.sol
SHA3: 45e2b9ce459001e9226e5446a5cc0365fc4b2a9e51695a2ecbd008562608d37e

File: ./vault/contracts/managementFeeStrategies/ManagementFee.sol
SHA3: 83749b40c7c3bdfbacf5c1feac4dd3270744133ac32618fb790c0f3b81a4ad38

File: ./vault/contracts/managementFeeStrategies/ProfitManagementFee.sol
SHA3: 335233a5054b30e1955546c9d6d598374e4d769a4a27943f1acdf026dd9117eb

File: ./vault/contracts/managementFeeStrategies/storage/ManagementFeeStorage.sol
SHA3: 605800b46ea07b8ae147048f0fd022da272503e208a895ccd5eef3127945c068

File: ./vault/contracts/proxies/YieldsterVaultProxy.sol
SHA3: 4ca57fec2077e14c7896041e4b98e80c0ba330ff9de68958df0fba56d8890859

File: ./vault/contracts/proxies/YieldsterVaultProxyFactory.sol
SHA3: 8e2ee9f8146215794c510496bffd74a8325ebdb25e9d481abecc075a55a30d2d

File: ./vault/contracts/safeUtils/safeMinter.sol
SHA3: 2e9e28bc6b54c12711e098152a2512f5212719a45360531f9053bc876a25e038

File: ./vault/contracts/safeUtils/safeUtils.sol
SHA3: 954f1de902a4dda3964ab76342013f54d8cc352db24ec40688123ba8e217a8aa

File: ./vault/contracts/SDKFunction.sol
SHA3: 723c2895022c74a79fab04f0748f44bd8129425d02122e78106c15209f1c81c0

File: ./vault/contracts/smartStrategies/deposit/StockDeposit.sol
SHA3: 3e64af0e6f812220a4613a999a5343a8cc6832c6835f8f299207df48aa01d4ec

File: ./vault/contracts/smartStrategies/withdraw/StockWithdraw.sol
SHA3: e47b4e722aa484965a1ac0025d767446d7044d6dc64315df231903a0f9d29744

File: ./vault/contracts/storage/TokenBalanceStorage.sol
SHA3: ac0b20202f19c656ae4480580d69f93750332db855a64b15454fdca78a68df23

File: ./vault/contracts/storage/VaultStorage.sol
SHA3: d60fe02e9cfe4d43dac963c603e8d4c0edbab437a171cdd6f659a239c4a9e12d

File: ./vault/contracts/utils/HexUtils.sol
SHA3: e59209f9dcabcbeb97a41291935f1660ddf7d21396cfc757678161c0f1aa97fb

File: ./vault/contracts/YieldsterVault.sol
SHA3: cb6cad2353f6821686c90f52531b76a7def1d4b06a9a3e9c365faf8fdd289d7a

File: ./aps/contracts/APContract.sol
SHA3: 8f0a407f140aedcad4a4f6422fe936cc762a71cd66ca259d1c252809cde9413f

File: ./vault/contracts/SDKFunction.sol
SHA3: 5fb7da752286f24435768d66a4e894facc25ae05edd5ac38089de99b6feb9cb7

File: ./price_module/contracts/price/PriceModuleV4.sol
SHA3: bf26ebb2357f28b585a11b2d38f908680a8ade31ddc321f3ee3c13242775b9df

File: ./whitelist/contracts/Whitelist.sol
SHA3: ff61abe17d9f215a8e14611ea27d9ec13e2d34c716b135cca2ccf18b486c9a53

File: ./exchange/contracts/ExchangeRegistry.sol
SHA3: 65a410fa5f652d8befe24db81ba4fd10811488335f2cb21fa3502655053e8de4

Second review scope

Repository:
https://github.com/Yieldster-Framework/yieldster-contracts
Commit:
e2aa198d98fd622373acde9c378a4fd5384c2356
Technical Documentation: Yes - https://docs.google.com/document/d/1b4riPQvwaibi1P4IdLNnL13EZYV8wxJ2/
JS tests: Yes
Contracts:

File: ./contracts/common/MasterCopy.sol
SHA3: f1be8c2decf07b0a0d07132cc9b1faa43b315e1aaf299457cb56379e703d0bde

File: ./contracts/exchange/Exchange.sol
SHA3: 447228abf67e6193b19c5f7da3e2f27b6f2ca0111ee9797484fd0da7bbfbed92

File: ./contracts/exchange/ExchangeRegistry.sol
SHA3: b2fbe73eb0c374f47a05e9fbf2e4a4a2947f97f649f930a201dca576e75dda54

File: ./contracts/managementFeeStrategies/ManagementFee.sol
SHA3: 7ebce26c347de8bda1256146357d13aa964b7d327b932fb9618b23038cb8b920

File: ./contracts/managementFeeStrategies/ProfitManagementFee.sol
SHA3: 1e680a1e6a34f257c96f5b6d8381518b046c1df5e73e0e4c35785ac5233b91ee

File: ./contracts/managementFeeStrategies/storage/ManagementFeeStorage.sol
SHA3: 4a3fab8f5131d37b5dd386af16bd0b16ee75534230c347cc1deb064329df4a9d

File: ./contracts/proxies/YieldsterVaultProxy.sol
SHA3: 4ca57fec2077e14c7896041e4b98e80c0ba330ff9de68958df0fba56d8890859

File: ./contracts/proxies/YieldsterVaultProxyFactory.sol

```
SHA3: 9ef92ba8618bd15c4019f932bc8d681501ed30047562a20c1ded9b12ebe70f49

File: ./contracts/safeUtils/safeMinter.sol
SHA3: 826e4c7021d5793d3535965d6fe0e5b09247346569ddae2f98591eb296ab3941

File: ./contracts/safeUtils/safeUtils.sol
SHA3: 26505aaf52a8d8971997f342c30d1bd403a6e0f3603144d118297fcac78d2916

File: ./contracts/YieldsterVault.sol
SHA3: 661dc385852b3e1cc062c3d28d30f81767ae027b9bcbc09a6188d01349e4f089

File: ./contracts/aps/APContract.sol
SHA3: 009f72a271bd2ffb6608f911e6cc5a2711316e9c005a05e618751b6e1fa35586

File: ./contracts/SDKFunction.sol
SHA3: 504e2a3609a78659cbd9204f4f46d80cc8e8be995caf4aa0975c7542aa5d40b3

File: ./contracts/smartStrategies/deposit/StockDeposit.sol
SHA3: e87dedc5708e61bb533fde825d218b4ab8980ead6786243c3978458f0b293721

File: ./contracts/smartStrategies/withdraw/StockWithdraw.sol
SHA3: 9e5fef20e1c782d5559a1b921ee8a8ad55ce3ccf5c5663dab91187633f413e3e

File: ./contracts/storage/TokenBalanceStorage.sol
SHA3: 90c21a2035c24bc8046950552f65777f76bc269de1aaaa2137d0ddce36f194c9

File: ./contracts/storage/VaultStorage.sol
SHA3: a87edb77e1d662377430a13a7d0b045398cc19a301383b5e7546e2032700aa55

File: ./contracts/utils/HexUtils.sol
SHA3: 6babc3177e146102fa24a5ede9cfe998de4dd445221df9385fb71ecf58190516

File: ./contracts/priceModule/PriceModuleV4.sol
SHA3: 37845ffd6ea192cb8a560371d7515e56de6fe15268b14ea30790ee898ef13c60

File: ./contracts/whitelist/Whitelist.sol
SHA3: 0c8fc5dd47d96871d0d9a3f75040976cea469f63281b3f35301c3593807a3361
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided full documentation with functional requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **3** out of **10**. The code follows the official guidelines. Unit tests were provided, but 12 test cases failed.

Architecture quality

The architecture quality score is **10** out of **10**. The project has a clean and clear architecture and a well-configured development environment.

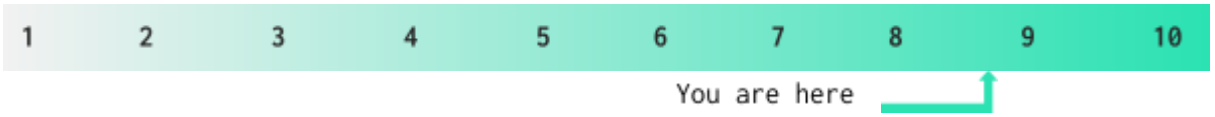
Security score

As a result of the audit, security engineers found **1** medium, and **7** low severity issues. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.6**



Notice

1. ERC20 tokens being used in the competitions should adopt standard ERC20 implementations. Any implementation with a fallback logic can lead to unexpected behavior during token transfers, such as reverts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Failed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Passed
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Failed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Tests Coverage	**Custom**	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

The Yieldster platform enables any developer to build blockchain applications by removing the learning curve. Using our end-to-end and easy-to-use solutions, developers can use the same programming language, development environment, and testing tools they are familiar with.

- **WhiteList Module:** Whitelisting is a process to filter the addresses that can deposit and withdraw assets in a vault. It can be used to blacklist the account addresses in the future.
- **Yieldster Vault:** Framework that follows an upgradable proxy pattern. It consists of smart contract snippets, templates, and tools that allow anyone to create yEarn-style vaults to design or choose a strategy, apply a strategy to the vault, decide who can deposit or use the vault and which assets & protocols are allowed in the vault. It provides a dashboard to track the progress, manage approved assets, and the whitelist groups. The vault is one of the core pieces of the liquidity balancer framework. It holds the assets against which a strategy can be applied and automates deposits and withdrawals as well as pay-outs and calculation of management fees.
- **YieldsterProxyModule:** Yieldster Vault proxy factory is used to create new proxies. The proxy factory provides a simple way to create a new proxy contract pointing to a master copy and executing a function in the newly deployed proxy all in one transaction. This additional transaction is generally used to execute the setup function to initialize the state of the contract.
- **Vault Storage:** Vault storage is used to hold the storage variables and frequently used functions by the Yieldster vault. It is used mainly to extend the features of a vault.
- **AP Contract:** A central contract that controls the entire framework along with its storage information.
- **Exchange Registry:** Exchange address is used to store the contract address which does the exchange conversion.
- **Exchange:** Exchange contract is used to perform the token conversion. Exchange inherits the vault storage like Yieldster vault, and the storage applicable to Yieldster vault is applicable to the exchange contract. There are delegate calls into the exchange contract so the contract can make the changes in other contracts.
- **Hex Utils**: Handles decimal conversion between tokens.
- **Stock Deposit:** This contract inherits the Vault Storage contract. By default, Stock deposit strategy.is implemented when a vault is created, even though there are many other deposit strategies,
- **Management Fee:** This contract (platform & strategy annual) is applied to all the vaults on creation. Management fee storage will have the platform fee %. It will calculate Nav of management fee accumulated during block difference and mint vault tokens corresponding to it for the Yieldster DAO. Management fee storage will have the strategy fee % for the respective strategies. It will calculate Nav of management

fee accumulated during block difference and mint vault tokens corresponding to it for the Strategy beneficiary.

- **Stock Withdraw**: This contract is used to withdraw assets from a vault.

Privileged roles

- Yieldster God is the address that can call emergency conditions of each vault.
- YieldsterDAO is the address that controls the complete Yieldster framework, the decision-maker of any change in the base protocols of the framework.
- YieldsterTreasury is the address where all the unsupported assets are deposited by the safeCleanup function.
- Beneficiary is the address that collects various annual fees collected from each strategy.
- Executor is a role that initiates instructions to the strategies.
- APS Manager: Each vault has its own APS Manager. The address sets the slippage and enables/ disables deposited and withdrawn assets in the vault.
- Whitelist manager is the address that can add/remove addresses and add/ remove whitelist manages to a whitelist group.
- Strategy Manager is the role inside a Vault that changes a strategy, such as activating/ deactivating a strategy and enabling/disabling protocols inside a strategy.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

1. **Use of undeclared identifier.**

   The `calculateFee` function which is used in `ProfitManagementFee` contract use undeclared variable `wEth`.

   This issue leads to the inability to compile the contracts.

   Contracts: vault/contracts/managementFeeStrategies/ProfitManagementFee.sol

   Function: calculateFee

   Recommendation: One of the possible solutions is to add a `wEth` argument to the `calculateFee` and pass it during the function call inside `executeSafeCleanUp` function.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

2. **Anyone can create a whitelist.**

   Whitelist creation allows anyone to create a whitelist. The whitelist logic checks if a user is any of the whitelists. This means that a malicious whitelist can be created.

   This can lead unauthorized parties to interact with the vaults.

   Contracts: Whitelist.sol

   Function: createGroup

   Recommendation: Apply access control to the whitelist creation logic.

   Status: Fixed (9b5720d6adade9fd6ee9b095d3fa87db1b7a058b)

3. **The contract allows transfer to Arbitrary Addresses.**

   SDKContract owner may withdraw vault funds to any contract. The `YieldsterVault` contract has the `protocolInteraction` function, which does a low-level call transaction to any contract specified in the `_poolAddress` argument.

   The owner of the contract may withdraw the funds from the vault instance.

   Contracts: vault/contracts/YieldsterVault.sol, vault/contracts/safe_utils/ safeUtils.sol

   Function: protocolInteraction, approvedAssetCleanup

   Recommendation: Pool address should be specified in the contract clearly.

   Status: Mitigated (The Customer stated that this is by design) + CHECK BOTH FUNCTION

4. **No function declaration.**

The `YieldsterVault` contract interacts with the `APContract`, but the `APContract` contract does not implement all the needed functions.

This may block functions execution inside the `YieldsterVault` contract.

Contracts:
vault/contracts/YieldsterVault.sol
aps/contracts/APContract.sol

Function: sdkContract

Recommendation: Add `sdkContract` function implementation.

Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

High

1. **Transfer can fail.**

   The `transfer` function is used to send ETH leftovers to a caller. If a sender is a contract with a fallback function, the execution will fail.

   It would be impossible to use a system by another contract.

   Contracts: StockWithdraw.sol, ProfitManagement.sol, ManagementFee.sol, SafeUtils

   Function: updateAndTransferTokens, transferFee, processEther, paybackExecutor

   Recommendation: use call() with a Gas limit to transfer value.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

2. **Reentrancy**

   While transferring ETH profit fees, the contract first transfers fees to the beneficiary and then updates the recorded ETH balance. When a malicious beneficiary is registered, an attacker can constantly call executeSafeCleanup to acquire more fees.

   This can lead to reentrancy vulnerabilities and cause the user to transfer more fees than expected.

   Contracts: ProfitManagement.sol, ManagementFee.sol

   Function: transferFee

   Recommendation: Re-implement function according to checks-effects-interactions pattern or use Reentrancy guard.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

3. **The contract can be updated at any time.**

   APContract implementation can be changed at any time, since it is an upgradeable contract. The user should have a chance to decide whether to continue to hold funds with the updated logic.

Contracts: APContract

Function: -

Recommendation: A time interval can be set between updates, so users can decide whether to continue with the new update.

Status: Mitigated. The deployer will be a DAO. The DAO will be handling the update. Therefore, it will have timelocks, so it cannot be updated frequently

Medium

1. **Checks-Effects-Interactions pattern violation.**

   Some contracts in the project update state variables after the interaction with ERC20 token.

   This can lead to unexpected behaviors in the function execution.

   Contracts:
   vault/contracts/managementFeeStrategies/ManagementFee.sol
   vault/contracts/managementFeeStrategies/ProfitManagementFee.sol
   vault/contracts/smartStrategies/deposit/StockDeposit.sol
   vault/contracts/smartStrategies/withdraw/Stockwithdraw.sol

   Function: transferFee, deposit, updateAndTransfer

   Recommendation: Follow the Checks-Effects-Interactions pattern. Update state variables before making external calls.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

2. **Function name functionality mismatch.**

   The setManagementFeeStrategies() function only adds new strategies; it does not allow to modify or remove existing strategies.

   The createVault() function does not create a new vault; it activates an existing one.

   Contracts: APContract.sol

   Function: setManagementFeeStrategies, createVault

   Recommendation: Rename aforementioned functions.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

3. **Potential Out-of-Gas exception.**

   Iterating over arrays (such as winners and bet slip lists) and making external calls may lead to enormous Gas consumption due to the arrays' size.

   Contracts:

   vault/contracts/managementFeeStrategies/ManagementFee.sol
   vault/contracts/managementFeeStrategies/ProfitManagementFee.sol
   vault/contracts/storage/VaultStorage.sol
   vault/contracts/exchange/Exchange.sol

vault/contracts/safeUtils/SafeUtils.sol

Function: getVaultNaV,exchangeSingleToken, executeSafeCleanup, processPlatformFee, approvedAssetCleanup

Recommendation: Implement array size limitations.

Status: Fixed(b5c73f40ba1abd15526a349e1e3e55c5889f06dc)

4. **Redundant conditional statement.**

   The `YieldsterVault` contract has the `onERC1155Received` function with a conditional statements which check the `id`. The function excecutes the same code blocks if `id == 2` and `id == 3`.

   This can lead to unnecessary Gas usage during the contract deployment.

   Contracts:
   vault/contracts/YieldsterVault.sol

   Function: onERC1155Received

   Recommendation: Join the `require` statement checks.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

5. **Functions duplication.**

   The `YieldsterVault` and `Exchange` contracts have functions for slippage calculations.

   This can lead to unnecessary Gas usage during the contract deployment.

   Contracts:
   vault/contracts/YieldsterVault.sol
   vault/contracts/exchange/Exchange.sol

   Function: calculateSlippage

   Recommendation: The `calculateSlippage` function gets data from the `APContract`, it is possible to make it `public` function.

   Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

6. **Unchecked delegateCall Return.**

   The return value Delegatecall made to another contract to execute cleanup is not being checked.

   Contracts:vault/contracts/safeUtils/SafeUtils.sol

   Function: managementFeeCleanup

   Recommendation: Implement return value checks.

   Status: Reported

7. **Wrong comments function description.**

There is a mismatch between NatSpec comments and argument in the `setWETH` function of the `APContract.sol`. The NatSpec comment contains a description for the unused parameter `_address`.

Contracts: aps/contracts/APContract.sol

Function: setWETH

Recommendation: Change the parameter name in the comment description to `_wEth`.

Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

Low

1. Unused event declaration.

The contract `APContract` declarates an event `VaultCreation`, but the event is never used.

Contracts: aps/contracts/APContract.sol

Function: addVault

Recommendation: The event should be emitted after the proper function is called.

Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

2. Floating Pragma.

The project uses floating pragma ^0.8.1.

Contracts: MasterCopy.sol, tokenBalanceStorage.sol, ExchangeRegistry.sol, APContract.sol, VaultStorage.sol, Exchange.sol, StockDeposit.sol, StockWithdraw.sol, ProfitManagementFee.sol, ManagementFee.sol, YieldsterVault.sol, SDKFunction.sol, Safeutils.sol, SafeMinter.sol

Function: -

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (bac8331300ffbf44b411ce2081775508c9f45d05)

3. Redundant use of SafeMath.

Since Solidity v0.8.0, the overflow/underflow check is implemented on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Contracts:

price_module/contracts/price/PriceModuleV4.sol
vault/contracts/YieldsterVault.sol
vault/contracts/exchange/Exchange.sol
vault/contracts/managementFeeStrategies/ManagementFee.sol
vault/contracts/managementFeeStrategies/ProfitManagementFee.sol

vault/contracts/safeUtils/SafeUtils.sol
vault/contracts/smartStrategies/deposit/StockDeposit.sol
vault/contracts/smartStrategies/withdraw/StockWithdraw.sol
vault/contracts/storage/VaultStorage.sol
vault/contracts/utils/HexUtils.sol

Recommendation: Remove SafeMath library.

Status: Reported

4. **Missing event emitting.**

Events for critical state changes (e.g. owner and other critical parameters) should be emitted for tracking things off-chain.

Contracts:
vault/contracts/YieldsterVault.sol
aps/contracts/APContract.sol
price_module/contracts/price/PriceModulev4.sol
vault/contracts/SDKFunction.sol
vault/contracts/YieldsterVault.sol
exchange/contracts/ExchangeRegistry.sol

Function:setVaultSlippage, changeVaultAdmin, setVaultSmartStrategy, setThreshold, setBeneficiaryAndPercentage, setInitialValues, addProxyFactory, addManager, setYieldsterGOD, setYieldsterDAO, setYieldsterTreasury, setEmergencyVault, changeCurveAddressProvider, setManager, changePriceModule, setAPS, transferOwnership, changeVaultAdmin, setVaultSmartStrategy, setYieldsterDAOTreasury, disableYieldsterGOD, setEmergencyVaul, setExchangeRegistry, setYieldsterExchange,addOrChangeSwapContrac

Recommendation: Create and emit related events.

Status: Reported

5. **Use of hard-coded values.**

Hard-coded values are used in computations for `slippage` calculations.

Contracts:
vault/contracts/YieldsterVault.sol
vault/contracts/exchange/Exchange.sol
vault/contracts/managementFeeStrategies/ProfitManagementFee.sol
vault/contracts/managementFeeStrategies/ManagementFee.sol
vault/contracts/safeUtils/SafeMinter.sol

Function: getVaultNAV, calculateSlippage, calculateFee, processPlatformFee, feeAmount, mintStragety

Recommendation: Create a precision constant and use it in the contracts.

Status: Reported

6. **Redundant use of low-level calls.**

The `YieldsterVault` contract has a function which uses a low level call to fire `changeMasterCopy` function.

Contracts: vault/contracts/YieldsterVault.sol

Function: upgradeMasterCopy

Recommendation: Update the authorization rule. Replace a low-level call with a direct function call.

Status: Reported

7. **Functions are used instead of modifiers.**

   The `YieldsterVault` widely used functions whereas modifiers may be used.

   Contracts: vault/contracts/YieldsterVault.sol

   Function: _isYieldsterGOD, _isVaultAdmin

   Recommendation: Replace functions with function modifiers `_isYieldsterGOD`.

   Status: Reported

8. **Custom owner functionality.**

   The contracts use custom owner functionality and do not use modifiers for access validation.

   Contracts:
   aps/contracts/APContract.sol
   vault/contracts/managementFeeStrategies/ManagementFee.sol
   price_module/contracts/price/PriceModuleV4.sol

   Recommendation: Use trusted implementations of Ownable contracts, for example, OpenZeppelin.

   Status: Reported

9. **Missing zero address validation.**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Contracts:
   price_module/contracts/price/PriceModulev4.sol
   vault/contracts/SDKFunction.sol
   vault/contracts/YieldsterVault.sol
   vault/contracts/safeUtils/SafeMinter.sol
   exchange/contracts/ExchangeRegistry.sol
   aps/contracts/APContract.sol

   Function: changeCurveAddressProvider, addToken, setManager, constructor, setSDKManager, changePriceModule, setAPS, transferOwnership, changeVaultAdmin, setVaultSmartStrategy, setBeneficiaryAndPercentage, setExecutor, initialize, setInitialValues, addProxyFactory, addManager, removeManager, setYieldsterGOD, setYieldsterDAO, setYieldsterDAOTreasury,

setEmergencyVault, setsafeUtils, setstringUtils, setExchangeRegistry, setYieldsterExchange, changeVaultAdmin, addOrChangeSwapContract

Recommendation: Implement zero address validations.

Status: Reported

10. Outdated Solidity version.

Using an old version prevents access to new Solidity security checks.

Contracts:
price_module/contracts/price/PriceModulev4.sol,
whitelist/contracts/Whitelist.sol

Function: -

Recommendation: Consider using one of these versions: *0.8.6, 0.8.9, 0.8.11*, or 0.8.13.

Status: Fixed (e2aa198d98fd622373acde9c378a4fd5384c2356)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.