HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SDAO
Date: September 30th, 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 SDAO
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	LP tokens system
Platform	EVM
Network	Ethereum, BSC
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://singularitydao.ai/
Timeline	01.09.2022 - 30.09.2022
Changelog	20.09.2022 - Initial Review 30.09.2022 - 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 SDAO (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/Singularity-DAO/DynasetForge
Commit:
29059274c4b78cb9ea85683129bdbdfcc5a8ea17
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/AbstractDynaset.sol
SHA3: bb447a424a0b10530f611b7efdf2a404c8938b3fe6b17471c11f86e3331810b6

File: ./contracts/AbstractDynasetFactory.sol
SHA3: 709a65edf023a278a86f48343b58ddfff7c33db1feda7e30958852808355d3e8

File: ./contracts/balancer/BConst.sol
SHA3: 26973f92b9ac2d1210d6d6cb6104e4037089503b88134e91134cdafd7d5054f4

File: ./contracts/balancer/BNum.sol
SHA3: ee5569a08d981eb403b31d1e41c1e8c39e4c0d8e0632f5ad9864dd476b203866

File: ./contracts/balancer/TToken.sol
SHA3: 556e32d39c987b25d512b8124cd7cc0128990239b48a8dd89a3321370ff094a3

File: ./contracts/DirectForge.sol
SHA3: c2d35e3f127eee86f5d3f51b6c14e97659f2e69d31106e8003a7f8dffdf52910

File: ./contracts/DToken.sol
SHA3: 3ae7232a52f4fdd452fe1ddf50ef514ba976bf4accaca55894eeb9f8248a4fc5

File: ./contracts/Dynaset.sol
SHA3: f8f216162057511305163b6bbfd7eeecfb9cae5df9f2d54d1f6e96746f0e56cf

File: ./contracts/DynasetFactory.sol
SHA3: e46f9e9d0eed3d83dad334a56ca97dae41c6164645f00c93f1439825922f7481

File: ./contracts/DynasetTvlOracle.sol
SHA3: eaef426688e9e7e0283dcbe0b82207a58bcf8fe231cbe19f9a63c900db8e8403

File: ./contracts/ForgeV1.sol
SHA3: 074a3b31bb44bc4718a517b14564ae9947a4c9cfff8ce062960eb58a61939930

File: ./contracts/interfaces/IDynaset.sol
SHA3: 91027c4ab1c489d8274ed70f5a5aae3084f10384e9a925c1e86a013307783353

File: ./contracts/interfaces/IDynasetContract.sol

```
SHA3: b2cbca9753dde72a7906436c2bde6644b29c63cf00ac814e6a1ae93bbf741726

File: ./contracts/interfaces/IDynasetForge.sol
SHA3: 6c3cd087b2e43fdea3c520097e7881bc6ea6dc970c35aacab46c70e442c5e68e

File: ./contracts/interfaces/IDynasetTvlOracle.sol
SHA3: 301d0f74b51123ebe998fbd8c329fea72e755ceebe5fbd8ac86567696fab81e8

File: ./contracts/interfaces/IERC20.sol
SHA3: d1a91dd8043db0ec6ac9713c037bce285771691e0927107eb580efc76a059301

File: ./contracts/interfaces/IUniswapV2Exchange.sol
SHA3: 7464fd0ab4db1a528f6f2eec224fc19c4fb519ecd9c80dde7293f6383c18a79a

File: ./contracts/interfaces/IUniswapV2Factory.sol
SHA3: aa3320eb5ed69447eddc917840e5b05cbeb879adbdd250dc62fd38ae34594d55

File: ./contracts/interfaces/IUniswapV2Pair.sol
SHA3: 494ba4054b5f2c0e85c0d9880e54d8c4262c2b699094bca9badc74e006ee24bb

File: ./contracts/interfaces/IUniswapV2Recipe.sol
SHA3: b410996459086beeb7bcb60b8db084e3e6553de8576407213266de51f7dabc29

File: ./contracts/interfaces/IUniswapV2Router.sol
SHA3: db9e2aea72ba9a85fe8cb17559d0e2f0a21270a3677316286ddbc6e1948f3217

File: ./contracts/interfaces/IUsdcOracle.sol
SHA3: 027aafdd93923a514b03924d4e038e86946d73eea49a99b242d589109168806d

File: ./contracts/interfaces/IWETH.sol
SHA3: 275bb193fe14d8ea3ef54f2fed5bb2aadf87825f8105e6a848e6079bec78d5c6

File: ./contracts/interfaces/OneInchAggregator.sol
SHA3: 51ebe65bc772829ea4969395f69fd92c1cb76065c75611f40ffc65b395de964d

File: ./contracts/libs/FixedPoint.sol
SHA3: aafca1030d47441a7cc9a26b1aaa5bd5578b322cd3aaf0cce4e285e871514aef

File: ./contracts/libs/OneInchSwapValidation.sol
SHA3: 6155b9563c07024bc1dd54519eea4a0653d6768ba35bb0740145edb5149a8cc0

File: ./contracts/libs/PriceLibrary.sol
SHA3: 7395b7903ed9476bad272281db50358c5519fe3fba57d5596c73ae1646bb41fb

File: ./contracts/libs/UniswapV2OracleLibrary.sol
SHA3: 5a6ffaa735bd714e694868d62c25a03f26e2a9073fb16010ce92bdc3235179dc

File: ./contracts/Migrations.sol
SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d

File: ./contracts/oracles/ChainlinkOracle.sol
SHA3: be1b49dd8a2e102ac7f4741a15ea1d2ac29770b06f58cb95320eae4df2a3a0f0

File: ./contracts/oracles/Uniswapv2Oracle.sol
SHA3: 6909952d6a4c7e5a2d87fa096341ac04aae029690d1c211d3453ced7d2e3b8ba

File: ./contracts/oracles/Uniswapv3Oracle.sol
SHA3: 2198a6dddf9b531d4b37229c8036bd8e2b12cd2a265e0213af6d2d8a794a5e53

File: ./contracts/oracles/UsdcOracle.sol
```

SHA3: c31262bb84283e8dfa3a28d34a64a72dbf65cdc631c5f8955efa716c5e941e31

File: ./contracts/recipe/LibSafeApproval.sol
SHA3: 8e26105ee69684b0de75443acf45e555417e430feedd3d1811f76d3fa2e4de3b

File: ./contracts/recipe/UniswapV2Library.sol
SHA3: fae0a20c4f7e281c08cb7b40e4140b54bf609e2d51c9f38c07b02797acbada79

Second review scope

Repository:
https://github.com/Singularity-DAO/DynasetForge
Commit:
58d76819f83b4aefecb43d889677807667ac3fcc
Documentation:
Functional requirements

Functional requirements
Integration and Unit Tests: Yes
Contracts:
File: ./contracts/AbstractDynaset.sol
SHA3: d08dcc63a38ba715bd85afe69c981c55f0057b80ca641770444801e6c9ced3e8

File: ./contracts/AbstractDynasetFactory.sol
SHA3: 8dbd0715ab4e0c8a16b95e77dd086152b9dd9208ffb40d3da7335d49a8805473

File: ./contracts/balancer/BConst.sol
SHA3: 26973f92b9ac2d1210d6d6cb6104e4037089503b88134e91134cdafd7d5054f4

File: ./contracts/balancer/BNum.sol
SHA3: ee5569a08d981eb403b31d1e41c1e8c39e4c0d8e0632f5ad9864dd476b203866

File: ./contracts/balancer/TToken.sol
SHA3: 556e32d39c987b25d512b8124cd7cc0128990239b48a8dd89a3321370ff094a3

File: ./contracts/DirectForge.sol
SHA3: b9adb450fab0ba0a51fca38cc13ddfb9eb119ba1d4c4f17b52ad49a79795a994

File: ./contracts/DToken.sol
SHA3: 3ae7232a52f4fdd452fe1ddf50ef514ba976bf4accaca55894eeb9f8248a4fc5

File: ./contracts/Dynaset.sol
SHA3: fd32ad31f84a1a709dc7f9c0e177e16b30e99ab9660c5bb21044b6908f566b11

File: ./contracts/DynasetFactory.sol
SHA3: e46f9e9d0eed3d83dad334a56ca97dae41c6164645f00c93f1439825922f7481

File: ./contracts/DynasetTvlOracle.sol
SHA3: 517ae27466f107d7d1c7d8ebb22d8c7cae17c07f5fce94087ac1ef491f8dad16

File: ./contracts/ForgeV1.sol
SHA3: 71b4444233aa29e18607bc80e27ac02fb8c9a72d522b50ba1f053547eb52e140

File: ./contracts/interfaces/IDynaset.sol
SHA3: 91027c4ab1c489d8274ed70f5a5aae3084f10384e9a925c1e86a013307783353

File: ./contracts/interfaces/IDynasetContract.sol
SHA3: b2cbca9753dde72a7906436c2bde6644b29c63cf00ac814e6a1ae93bbf741726

File: ./contracts/interfaces/IDynasetTvlOracle.sol

```
SHA3: 301d0f74b51123ebe998fbd8c329fea72e755ceebe5fbd8ac86567696fab81e8

File: ./contracts/interfaces/IERC20.sol
SHA3: d1a91dd8043db0ec6ac9713c037bce285771691e0927107eb580efc76a059301

File: ./contracts/interfaces/IUniswapV2Pair.sol
SHA3: 494ba4054b5f2c0e85c0d9880e54d8c4262c2b699094bca9badc74e006ee24bb

File: ./contracts/interfaces/IUniswapV2Router.sol
SHA3: db9e2aea72ba9a85fe8cb17559d0e2f0a21270a3677316286ddbc6e1948f3217

File: ./contracts/interfaces/IUsdcOracle.sol
SHA3: f7ee8bf6d7f3e01650e0843f435b4e830481a0a1038aa1f0d699516f58b45373

File: ./contracts/interfaces/IWETH.sol
SHA3: 275bb193fe14d8ea3ef54f2fed5bb2aadf87825f8105e6a848e6079bec78d5c6

File: ./contracts/interfaces/OneInchAggregator.sol
SHA3: 51ebe65bc772829ea4969395f69fd92c1cb76065c75611f40ffc65b395de964d

File: ./contracts/libs/FixedPoint.sol
SHA3: aafca1030d47441a7cc9a26b1aaa5bd5578b322cd3aaf0cce4e285e871514aef

File: ./contracts/libs/OneInchSwapValidation.sol
SHA3: 6155b9563c07024bc1dd54519eea4a0653d6768ba35bb0740145edb5149a8cc0

File: ./contracts/libs/PriceLibrary.sol
SHA3: 7395b7903ed9476bad272281db50358c5519fe3fba57d5596c73ae1646bb41fb

File: ./contracts/libs/UniswapV2OracleLibrary.sol
SHA3: 5a6ffaa735bd714e694868d62c25a03f26e2a9073fb16010ce92bdc3235179dc

File: ./contracts/Migrations.sol
SHA3: 1ed21175afe224f2dec4194e8acc3c1c2a8f88df513f6f81a4a578b9b487d53d

File: ./contracts/oracles/ChainlinkOracle.sol
SHA3: 13662629aa016b5b027bfc434a157fc78ff5c6439ea7bfcf266d90baa0ad63e4

File: ./contracts/oracles/Uniswapv2Oracle.sol
SHA3: 6797d58792a3b85ec1d99f08f68995e241b71bd852c740e5544b930457ec7f70

File: ./contracts/oracles/Uniswapv3Oracle.sol
SHA3: 363cdc6ef4b26054ad2fb4c9a32a13594c562177b42eacaefa6419db632a80b8

File: ./contracts/oracles/UsdcOracle.sol
SHA3: f16ae90bc00c408507a1b812a0225b6048fcdb9ce54512006ef34cdfcef2087a

File: ./contracts/recipe/UniswapV2Library.sol
SHA3: fae0a20c4f7e281c08cb7b40e4140b54bf609e2d51c9f38c07b02797acbada79
```

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. Functional requirements are provided. A technical description is provided as comments in the code.

Code quality

The total Code Quality score is **7** out of **10**. Redundant declarations and code duplications were found. Tests were provided. Integration Hardhat tests coverage is **60%**; **14%** for Hardhat unit tests coverage, and **18%** for Truffle tests.

Architecture quality

The architecture quality score is **10** out of **10**. The Truffle is provided as a development environment; it implements deployment scripts and tests (The Hardhat is used for tests as well.)

Security score

As a result of the audit, the code contains **1** medium and **3** 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: **9**.

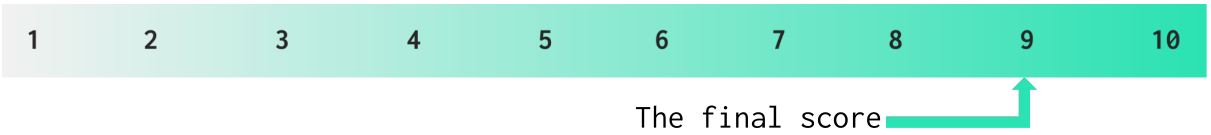

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
19 September 2022	14	3	4	3
30 September 2022	3	1	0	0

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.	Not Relevant
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 self-destructible while 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
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
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.	Passed
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
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
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 or be predictable.	Not Relevant
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.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
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	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
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
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Dynaset is an actively managed multi-asset on-chain crypto investment system with the following contracts:

- *AbstractDynaset* – is an abstract contract with the base Dynaset functionality. The contract is an ERC-20 token. Token`s name and symbol are defined when the contract deployment, the total supply is unlimited. The contract allows depositing tokens in appropriate weights for each token to get the desired amount of the Dynaset tokens. Resulting Dynaset share is calculated according to the sum of UDS prices of deposited tokens to the total deposited tokens in USD ratio (Uniswap oracle). The Dynaset tokens can be redeemed: the appropriate amounts of the tokens in the pool will be transferred to the user during the redeeming, and the Dynaset tokens will be burnt. The addresses should be allowed to deposit and redeem tokens.
- *Dynaset* – is a contract that inherits the *AbstractDynaset* contract. The functionality of the contract allows the user with the *digitalAssetManager* role to swap the pool tokens using the Uniswap and 1inch.
- *AbstractDynasetFactory* – is an abstract contract with the base DynasetFactory functionality for managing *Dynaset* contracts (*Dynaset*s creation is not implemented in this contract).
  The functionality allows the owner to collect fees from the *Dynaset*:
  - Performance fee: the fee is the defined percentage of the total amount of tokens in the contract in USD (up to 25%, is defined for each *Dynaset* contract when initialization). The fee collected is transferred to the contract in USD tokens. Fee collecting can be triggered once a month.
  - Management fee: the fee is the defined percentage from the year's total amount of tokens in the contract in USD (up to 5%, is defined for each *Dynaset* contract when initialization): the amount of calculated fee is multiplied by the amount of time since the last fee collection and is divided by the year time. Fee collecting can be triggered once a month, together with the performance fee.

  The owner can withdraw tokens from the contract to the defined deployment address. The owner can create a snapshot for each *Dynaset*: the total amount of tokens in the contract in USD at that moment will not be considered in the fee calculations. The owner sets and updates the oracles (*DynasetTvlOracle*) for each *Dynaset* that are used for the fee calculations. The fees are transferred to the defined *gnosis* address.
- *DynasetFactory* – is a contract that inherits the *AbstractFactory* contract and allows the creation of *Dynasets.*
- *ForgeV1* – is a contract that allows depositing liquidity for the *Dynasets* and redeeming tokens for *Dynasets.* The contract admin (user

with the *BLACK_SMITH* role*)* sets the forges; each forge object has the following limits and properties:
  - The contribution token - token that is contributed by users (it may be ETH);
  - Minimal and maximal amount that can be contributed by the user;
  - Maximal capital of total deposited tokens;

  The user with the *BLACK_SMITH* role triggers the forging start, and tokens deposing stops. While forging, the contribution token is swapped through the Uniswap for the *Dynaset* underlying tokens according to the USD tokens ratios. The *Dynasets* are minted for users according to the deposited amounts to the *ForgeV1* contract. Users can withdraw *Dynasets* or redeem them.

  Redeeming: the output *Dynasets* tokens amounts are swapped through Uniswap to the defined by the user token (it should be one of the underlying *Dynasets* tokens), and transferred to the user with the charged fee. The fee is calculated according to the forging period (duration between current time and the time where the forge object was created):
  - 0 - 30 days: 5%
  - 31 - 60 days: 4%
  - 61 - 90 days: 2.5%
  - above 91 days: 0%

  The functionality allows users to directly deposit Dynasets for their following redeeming.
- *DirectForge* – is a contract that allows directly depositing liquidity for the Dynasets and redeeming tokens for Dynasets. (User with the *BLACK_SMITH* role allows and stops the depositing and redeeming). The fees are taken when redeeming in the same amounts as in the *ForgeV1* contract.
- *DynasetTvlOracle* – is a contract that interacts with oracle (*UsdcOracle* contract), obtains USD prices for Dynasets and underlying tokens, allows to update tokens prices, used in the *AbstractDynaset*, *AbstractDynasetFactory*, *DirectForge* and ForgeV1 contracts.
- *UniswapV2Library* – is a library with helper methods for the Uniswap oracle, used in the *Uniswapv2Oracle* and *PriceLibrary* contracts.
- *PriceLibrary* – is a library that helps to retrieve and aggregate data from the Uniswap oracle, used in the *Uniswapv2Oracle* contract.
- *FixedPoint* – is a library for handling binary fixed-point numbers, used in the *Uniswapv2Oracle*, *PriceLibrary*, *UniswapV2OracleLibrary* contracts.
- *UniswapV2OracleLibrary* – is a library with helper methods for the Uniswap oracle, used in the *PriceLibrary* contract.
- *OneInchSwapValidation* – is a library with helper methods for validating Uniswap and 1inch swap pools used in the *Dynaset* contract.

- *LibSafeApprove* – is a library with a safe approve function.
- *DToken* – is an ERC-20 token contract, inherited by the *AbstractDynaset* contract.
- *DTokenBase* – is a contract with base ERC-20 token functional, inherited by the *DToken* contract.
- *BConst* – is a helper contract that contains constants, used in the *BNum* contract.
- *BNum* – is a contract that contains the functional for calculations, used in the *AbstractDynaset* contract.
- *UsdcOracle* – is a contract that interacts with the oracles, obtains prices and allows to update oracles. The contract contains the preferred oracle; the user with the *ORACLE_ADMIN* role adds fallback oracles to the contract; when obtaining prices, they are requested from all the oracles in turn until the stale (2 days initially, can be changed by the user with the *ORACLE_ADMIN* role) price is returned.
- *Uniswapv2Oracle* – is a contract that allows getting prices of tokens from the Uniswap V2 oracle. The contract provides average token prices that are measured using the recent and historical prices.

  The prices for tokens can be obtained after the defined deployment time *periods*. All the prices are stored in the contract state (in USDC for WETH, and in the WETH for all the other tokens). The average token prices are measured using the current token price received from the oracle and the previously obtained one if it is at least half a *period* older than now and at most 2 *periods* older.
- *Uniswapv3Oracle* – is a contract that allows getting prices of tokens from the Uniswap V3 oracle.
- *ChainlinkOracle* – is a contract that allows getting prices of tokens from the Chainlink oracle.
- *IDynaset* – is an interface for the *Dynaset* contract, used in the *DirectForge, DynasetTvlOracle, ForgeV1* contracts.
- *IDynasetContract* – is an interface for the *Dynaset* contract, inherited by the *AbstractDynaset,* used in the *AbstractDynasetFactory* contract.
- *IDynasetTvlOracle* – is an interface for the *DynasetTvlOracle* contract, inherited by the *DynasetTvlOracle* contract, used in the *AbstractDynaset, AbstractDynasetFactory, DirectForge, DynasetTvlOracle, ForgeV1* contracts.
- *IERC20* – is an interface for the ERC-20 tokens, inherited by the *DToken*, *IDynaset, IWETH* contracts, used in the *LibSafeApprove* contract.
- *IUniswapV2Pair* – is an interface for Uniswap pair, used in the *AbstractDynaset*, *OneInchSwapValidation*, *UniswapV2OracleLibrary*, *UniswapV2Library* contracts.

- IUniswapV2Router – is an interface for Uniswap router, used in the *AbstractDynaset*, *DirectForge*, *Dynaset*, *ForgeV1* contracts.
- *IUsdcOracle* – is an interface for USDC oracle, inherited by the *UsdcOracle*, *Uniswapv3Oracle*, *Uniswapv2Oracle*, *ChainlinkOracle* contract, used in the *DynasetTvlOracle* contract.
- *IWETH* – is an interface for WETH, used in the *ForgeV1* contract.
- *OneInchAggregator* – is an interface for the 1INCH aggregator, used in the Dynaset contract.
- *IDynasetForge, IUniswapV2Exchange, IUniswapV2Factory, IUniswapV2Recipe* – are the interfaces not used in the project.

Privileged roles

- The *AbstractDynaset* and *Dynaset* contracts have privileged roles of controller, factory and digitalAssetManager:
  - The controller can set addresses that can deposit and redeem tokens.
  - The factory can initialize the contract, withdraw fees and set the oracle address.
  - The digitalAssetManager can add and remove pool tokens (remove when the token balance in the contracts is 0).
- The *Dynaset* contract has privileged roles of controller, factory and digitalAssetManager:
  - The digitalAssetManager can set the deadline period that is used for Uniswap swaps, and update the Uniswap v2 Router address.
  - The controller can swap the pool tokens using the Uniswap and 1inch.
- The owner of the *AbstractDynasetFactory* and *DynasetFactory* contract can initialize *Dynaset* contracts, collect and withdraw fees from them, set and update *DynasetTvlOracle* contracts for *Dynaset* contracts, update the gnosis address (address where the fees are transferred to).
- The owner of the *DynasetFactory* contract can create *Dynaset* contracts.
- The *BLACK_SMITH* role of the *ForgeV1* contract allows to create forging objects, to trigger forging, to set status of *Dynaset* and underlying tokens withdrawal and underlying tokens depositing: allowed or not, to withdraw fees for redeeming, set Uniswap address, deadline and slippage, oracle address.
- The *BLACK_SMITH* role of the *DirectForge* contract allows to set status of Dynaset and underlying tokens withdrawal and underlying tokens depositing: allowed or not, to withdraw fees for redeeming, set the redeem start date (used for fees calculations), set Uniswap address, deadline and slippage, oracle address.

- The *ORACLE_ADMIN* role of the *UsdcOracle* contract allows to set fallback oracles and stale price period, and pause the oracles.
- The *ORACLE_ADMIN* role of the *Uniswapv3Oracle* contract allows to set the Uniswap fee.

Risks

- In the *Uniswapv2Oracle* contract, the average prices are computed using the current and previously manually obtained token prices; they should have the correct time gap between each other. Therefore, it is impossible to get the average prices when the token price has been obtained the first time and the time period has not passed, or when the token price has not been updated in time and the required time period has expired. (https://docs.uniswap.org/protocol/V2/guides/smart-contract-integration/building-an-oracle#oracle-maintenance)
- Despite the documentation specifying that there are AI contracts to manage the funds based on the profit forecasts, the project does not contain any AI contracts. (It is impossible to run AI on the blockchain)
- If the calculated fee amount in the *DynasetFactory* is greater than the USD balance on the *Dynaset* contract, it would be impossible to collect the fee.

Findings

■■■■ Critical

1. Denial of Service vulnerability

When obtaining the prices from the *fallbackOracles*, for the *price* and *observation* values assignment, the *value > value* condition is checked, which always returns *false*.

Therefore, the prices from *fallbackOracles* can not be obtained.

Path: ./contracts/oracles/UsdcOracle.sol : getPrice()

Recommendation: replace the *value > value* condition with *value > 0*, or fix the logic in another required way, and ensure that prices from *fallbackOracles* can be obtained.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

2. Denial of Service vulnerability

Functions *updateTokenPrices* of *ChainlinkOracle* and *Uniswapv3Oracle* do not update tokens prices, and *canUpdateTokenPrices* functions return *false* value.

Therefore, these oracles are inoperable and cannot provide prices for *UsdcOracle* contract.

Paths: ./contracts/oracles/ChainlinkOracle.sol : canUpdateTokenPrices(), updateTokenPrices(); ./contracts/oracles/Uniswapv3Oracle.sol : canUpdateTokenPrices(), updateTokenPrices()

Recommendation: ensure that *ChainlinkOracle* and *Uniswapv3Oracle* contract can provide tokens prices for the *UsdcOracle* contract.

Status: Mitigated (The Customer comment: “Works as designed. Might have been better to name canUpdateTokenPrices to needsToUpdateTokenPrices. Since chainlink and uniswapv3 oracles are updates by 3rd party actors. The oracles do provide prices even when we don't update them explicitly.”)

3. Flashloan attack

Uniswapv3Oracle and Uniswapv2Oracle contracts use Uniswap router for the prices obtaining.

The prices in the Uniswap may be disbalanced using the flashloan, and the price may be manipulated.

Paths: ./contracts/oracles/Uniswapv3Oracle.sol, ./contracts/oracles/Uniswapv2Oracle

Recommendation: do not the current price for the price calculation, replace the arithmetic mean with the geometric mean in the *Uniswapv2Oracle*.

Status: Mitigated (The Customer comment: “The oracles are taking the average price of two observations at least a minimal observation period in the past. Which means an attacker needs to manipulate the price over several minutes without it getting arbitraged. A flashloan has zero effect on such oracles.”)

■■■ High

1. Requirements violation

According to the documentation, users with the *BLACK_SMITH* role of the *ForgeV1* and *DirectForge* contracts should be able to set the withdrawal fee. However, the fee is calculated dynamically according to the forging period. (*capitalSlash* function)

Therefore, the code does not match the requirements.

Path: ./contracts/ForgeV1.sol : capitalSlash()

Recommendation: implement the code according to the requirements or change the requirements to match the code.

Status: Mitigated (The Customer comment: “Specificed in the documentation already, this is an Exit Fee on the redeem function and not linked to the performance & managements fees (that can be executed by the BLACKSMITH)”)

2. Undocumented functionality

The functionality allows user with *BLACK_SMITH* role to define if users may redeem or withdraw dynasets. Such functionality is not described in the documentation.

This may block the withdrawal of users` funds.

Path: ./contracts/ForgeV1.sol : setWithdraw(), setlpWithdraw()

Recommendation: remove the undocumented functionality or describe it in the documentation.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

3. Requirements violation; Denial of Service vulnerability

Function *updateTokenPrices* runs over *fallbackOracles* array to update tokens prices, but the tokens for the *preferredOracle* are not updated.

Therefore, tokens may be un-updated or not updated in time, leading to the inoperability of the oracle.

Path: ./contracts/oracles/UsdcOracle.sol : updateTokenPrices()

Recommendation: ensure that all the token prices are updated.

Status: Mitigated (The Customer comment: “Works as designed. Since chainlink oracle is used as preferred oracle. The oracles do provide prices even when we don't update them explicitly. Even when the prices are not updated in the preferred oracle, it will return the fallback oracle.”)

4. **Unsecure oracles usage**

   The *UsdcOracle* contract does not allow to remove oracles it relies on (*preferredOracle*, *fallbackOracles*).

   Therefore, if the oracle is compromised, it will be impossible to pause it.

   Path: ./contracts/oracles/UsdcOracle.sol : preferredOracle, fallbackOracles

   Recommendation: add the ability to pause oracles.

   Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

Medium

1. **Denial of Service vulnerability**

   The *getDepositors* function runs over the depositors array.

   If the array with depositors is large enough to make the Gas required for executing the for loop exceed the block Gas limit, the function may be inoperable.

   Path: ./contracts/ForgeV1.sol : getDepositors()

   Recommendation: fix the logic not to rely on array length.

   Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

2. **Tests failing**

   48 Truffle and 2 Hardhat integration tests are failing.

   Recommendation: ensure that all tests are passing.

   Status: Reported

3. **Checks-Effects-Interaction pattern violation**

   The state variables are changed after the external calls.

   This may lead to re-entrants and race conditions.

   Path: ./contracts/ForgeV1.sol : forgeFunction(), deposit(), redeem()

Recommendation: implement the code according to the Checks-Effects-Interaction pattern or use nonReentrant modifier.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

Low

1. Boolean equality

The values are compared to *true* and *false* instead of a direct boolean check.

Paths: ./contracts/AbstractDynaset.sol : joinDynaset(), exitDynaset(); ./contracts/ForgeV1.sol : forgeFunction()

Recommendation: remove the boolean equality.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

2. Code duplication

The code is duplicated for depositing tokens for *if (forge.isEth)* and *else* conditions.

This may lead to unnecessary Gas consumption.

Path: ./contracts/ForgeV1.sol : deposit()

Recommendation: replace the common code into separate function.

Status: Mitigated (The Customer comment: “Code logic is different and it is not worth splitting the common code into a function”)

3. Misleading variable name

The *amountInMax* array stores the minimal output amounts for the swap.

Misleading names of variables decrease the code readability and may lead to errors in code understanding.

Path: ./contracts/ForgeV1.sol : redeem()

Recommendation: ensure that the variable name indicates the value it stores.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

4. Redundant functionality

The internal functions use *onlyRole(BLACK_SMITH)* modifier, but the same modifier is used in the external functions where these internal functions are used.

This may lead to unnecessary Gas consumption.

Path: ./contracts/ForgeV1.sol : _setDeposit(), _setWithdraw(), _createForge()

Recommendation: remove the redundant modifiers.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

5. **Redundant role**

The *ORACLE_ADMIN* roles of the *DynasetTvlOracle*, *Uniswapv2Oracle*, *ChainlinkOracle* contracts are never used.

Paths: ./contracts/DynasetTvlOracle.sol : ORACLE_ADMIN; ./contracts/oracles/Uniswapv2Oracle.sol : ORACLE_ADMIN; ./contracts/oracles/ChainlinkOracle.sol : ORACLE_ADMIN;

Recommendation: remove the redundant roles.

Status: Reported

6. **Never used functions**

There are never used functions in the contracts.

Unused code decreased the code readability.

Paths: ./contracts/libs/FixedPoint.sol : encode(), encode144(), div(), decode(); ./contracts/balancer/BNum.sol : bpow(), bpowApprox(), bdiv(), bsubSign(), badd(), bsub(), bpowi(), bfloor(), btoi(); ./contracts/ForgeV1.sol : _burnDynaset()

Recommendation: remove never used functions.

Status: Reported

7. **Never used library**

The project contains a never used library.

Path: ./contracts/recipe/LibSafeApproval.sol

Recommendation: remove the redundant library.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

8. **Never used variables**

The contract contains variables that are never used.

Unused code decreased the code readability.

Path: ./contracts/balancer/BConst.sol : WEIGHT_UPDATE_DELAY, WEIGHT_CHANGE_PCT, MIN_FEE, MAX_FEE, EXIT_FEE, MAX_IN_RATIO, MAX_OUT_RATIO

Recommendation: remove the redundant code.

Status: Reported

9. Outdated and floating pragma

The contract uses the `>=0.4.22 <0.9.0` pragma.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version.

Path: ./contracts/Migrations.sol

Recommendation: use a contemporary and locked compiler version.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

10. Unused interfaces

The project contains never used interfaces.

Paths: ./contracts/interfaces/IDynasetForge.sol, ./contracts/interfaces/IUniswapV2Exchange.sol, ./contracts/interfaces/IUniswapV2Factory.sol, ./contracts/interfaces/IUniswapV2Recipe.sol

Recommendation: remove unused interfaces.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

11. Redundant imports

The contracts contain imports that are not used.

Unused code decreases the code readability.

Paths: ./contracts/AbstractDynasetFactory.sol : "@openzeppelin/contracts/utils/math/SafeMath.sol"; ./contracts/AbstractDynaset.sol : "./interfaces/IUniswapV2Pair.sol", "./interfaces/OneInchAggregator.sol"

Recommendation: remove the redundant imports.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

12. Default visibility usage

There is variable default visibility usage.

The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

Path: ./contracts/oracles/UsdcOracle.sol : staleOraclePeriod

Recommendation: define the visibility explicitly.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

13. Block values as a proxy for time

The contracts use *block.timestamp* for time calculations.

It is not precise and safe.

Paths: ./contracts/ForgeV1.sol : redeem(), capitalSlash(), _mintDynaset(), _createForge(); ./contracts/DynasetFactory.sol : deployDynaset(); ./contracts/Dynaset.sol : swapUniswap(); ./contracts/DirectForge.sol : instantMint(), instantRedeem(), capitalSlash()

Recommendation: it is recommended to avoid using *block.timestamp*. Alternatively, it is safe to use oracles.

Status: Mitigated (The Customer comment: “There is not a lot of impact by using the block.timestamp in our contracts. There is also no viable alternative for the logic we need.”)

14. Missing zero addresses validations

The parameters *tokens* and *tokenProvider* are not checked for a non-zero value.

This can lead to unwanted external calls to 0x0.

Path: ./contracts/AbstractDynaset.sol : initialize

Recommendation: add the zero address checks.

Status: Fixed (Revised commit: 58d76819f83b4aefecb43d889677807667ac3fcc)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.