NXD LPGateway Audit
Security Audit Report

PREPARED FOR:

NXD Protocol

ARCADIA CONTACT INFO

Email: audits@arcadiamgroup.com

Telegram: https://t.me/thearcadiagroup

Revision history

Date	Reason	Commit
05/10/2024	Initial Audit Scope	#91d60a134b6a8ca52e27356b70815e4c8c549dc2
06/16/2024	Review of remediations	#6382e5629263954b4be2c98abff3c9da525d783e

Table of Contents

Executive Summary

1. Introduction and Audit Scope

NXD Protocol engaged Arcadia to perform a security audit of their core protocol smart contracts, our review of their codebase occurred in the repo **DXNhyperstructure/NXD-Protocol** on the commit hash **#91d60a134b6a8ca52e27356b70815e4c8c549dc2**

a. Review Team

Van Cam Pham - Lead Security Engineer

b. Project Background

The **NXD Protocol**, featuring the NXD cryptocurrency, is designed as a hyper-deflationary asset that serves as a derivative of the DXN Protocol's ROI. It incorporates an automated vault system that enhances its daily earnings through various strategies aimed at continuously burning NXD tokens. This process is intended to decrement the token's supply while augmenting its baseline price. Additionally, the protocol leverages its operating profits by repurchasing and staking DXN tokens. This action not only amplifies the compounding of DXN but also boosts the Ethereum (ETH) rewards necessary to support the protocol's deflationary mechanics, thereby ensuring a sustainable model for its economic ecosystem.

c. Coverage

For this audit, we performed research, test coverage, investigation, and review ofthe following code repositories, files, and/or libraries are considered in scope for the review.

File
src/QDistributor.sol
src/LPGateway.sol
test/LPGateway.t.sol

2. Audit Summary

a. Audit Methodology

Arcadia completed this security review using various methods, primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

The followings are the steps we have performed while auditing the smart contracts:

- Investigating the project and its technical architecture overview through its documentation
- Understanding the overview of the smart contracts, the functions of the contracts, the inheritance, and how the contracts interface with each other thanks to the graph created by Solidity Visual Developer
- Manual smart contract audit:
    - Review the code to find any issue that could be exploited by known attacks listed by Consensys
    - Identifying which existing projects the smart contracts are built upon and what are the known vulnerabilities and remediations to the existing projects
    - Line-by-line manual review of the code to find any algorithmic and arithmetic related vulnerabilities compared to what should be done based on the project's documentation

    - Find any potential code that could be refactored to save gas
    - Run through the unit-tests and test-coverage if exists
- Static Analysis:
    - Scanning for vulnerabilities in the smart contracts using Static Code Analysis Software
    - Making a static analysis of the smart contracts using Slither
- Additional review: a follow-up review is done when the smart contracts have any new update. The follow-up is done by reviewing all changes compared to the audited commit revision and its impact to the existing source code and found issues.

b. Summary

There were **6** issues found, **0** of which were deemed to be 'critical', and **0** was rated as 'high'.

Severity Rating	Number of Original Occurrences	Number of Remaining Occurrences
CRITICAL	0	0
HIGH	0	0
MEDIUM	1	0
LOW	2	0
INFORMATIONAL	2	1
GAS	1	0

Findings in Manual Audit

1. Readonly state variables should be defined with constant

Issue ID

NXDP-1

Status

Resolved

#6382e5629263954b4be2c98abff3c9da525d783e

Risk Level

Severity: Low

Code Segment

```
// QDistributor.sol
address public nxdStakingVault = 0xa1B56E42137D06280E34B3E1352d80Ac3BECAF79;
address public nxdProtocol = 0xE05430D42842C7B757E5633D19ca65350E01aE11;

IUniswapV2Router02 public UNISWAP_V2_ROUTER =
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);

IUniswapV2Pair nxdDXNPair =
IUniswapV2Pair(0x98134CDE70ff7280bb4b9f4eBa2154009f2C13aC);
    IV2Oracle public v2Oracle =
IV2Oracle(0x14D558267A97c7a61554d7F7b23a594781E04495);

// LPGateway.sol
IUniswapV2Router02 public UNISWAP_V2_ROUTER =
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);

IUniswapV2Pair public dxnNXDPair =
IUniswapV2Pair(0x98134CDE70ff7280bb4b9f4eBa2154009f2C13aC);
```

Description

State variables that are readonly should be defined with the *constant* keyword for saving storage gas costs.

Code location

```
src/LPGateway.sol
src/QDistributor.sol
```

Reference

https://docs.soliditylang.org/en/v0.6.7/contracts.html?highlight=immutable#constant-and-immutable-state-variables

Recommendation

Define readonly state variables with the *constant* keyword.

2. Redundant approves waste gas

Issue ID

NXDP-2

Status

Resolved

#6382e5629263954b4be2c98abff3c9da525d783e

Risk Level

Severity: Gas

Code Segment

```
function addLiquidity(
        uint256 amountNXDDesired,
        uint256 amountDXNDesired,
        uint256 amountNXDMin,
        uint256 amountDXNMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountA, uint256 amountB, uint256 liquidity) {
        nxd.transferFrom(msg.sender, address(this), amountNXDDesired);
        dxn.transferFrom(msg.sender, address(this), amountDXNDesired);

        nxd.approve(address(UNISWAP_V2_ROUTER), type(uint256).max);
```

```
        dxn.approve(address(UNISWAP_V2_ROUTER), type(uint256).max);
        //...
}
```

Description

The function approves for the uniswap router contract to spend its token balance when adding liquidity. The function approves both tokens NXD and DXN at an amount of maximum value of uint256. When approving max for a token, all subsequent *approves* for the same spender are redundant, thus wasting gas.

Code location

```
src/LPGateway.sol
```

Recommendation

- Use a boolean state variable to indicate whether approvals have been done.
- If yes, all subsequent transactions for addLiquidity do not need token approvals.

3. Use custom errors instead of require statements

Issue ID

NXDP-3

Status

Resolved

#6382e5629263954b4be2c98abff3c9da525d783e

Risk Level

Severity: Informational

Description & Code Segment

Custom errors are available from Solidity. Instead of using error strings, to reduce deployment and runtime cost, you should use custom errors.

https://soliditylang.org/blog/2021/04/21/custom-errors/

Code location

```
src/LPGateway.sol
src/QDistributor.sol
```

Proof of concept

–

Recommendation

Consider replacing the *require* statements with `if (something) revert CustomError()` type of checks.

4. Make sure LPGateway contract address whitelisted

Issue ID

NXDP-4

Status

—

Risk Level

Severity: Informational

Code Segment

```
nxd.transferFrom(msg.sender, address(this), amountNXDDesired);
        dxn.transferFrom(msg.sender, address(this),
amountDXNDesired);
```

Description

- The *transferFrom* function of NXD contract is called to transfer NXD from sender to the LPGateway contract. As the token has tax on transfer, it is recommended to make sure the deployed LPGateway contract address is whitelisted for tax free.

Code location

```
src/LPGateway.sol
```

Recommendation

- Line 448: *The* balance of the NXD token contract should increase the amount set in *sellNXDAmount*, and the function should emit a *Transfer* event *from* from to the *NXD* contract
- Whitelisting the LPGateway deployed contract address for transfer tax free

5. Console.log should be removed.

Issue ID

NXDP-5

Status

Unresolved

Risk Level

Severity: Low

Description

Console.log is used for debugging purposes and should not be part of the production code.

Code Location

```
All scoped files
```

Recommendation

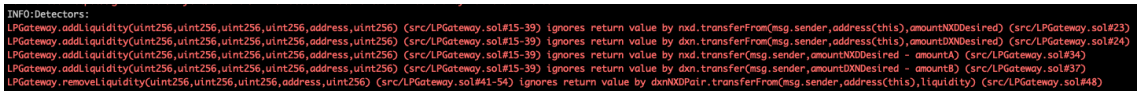
Remove console.logs

Automated Audit

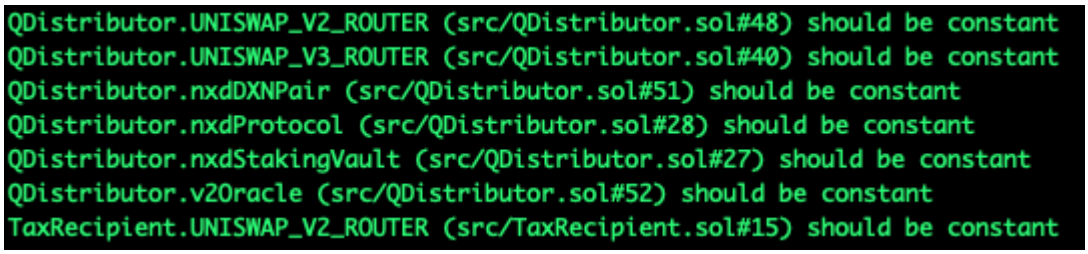
Static Analysis with Slither

The following shows the results found by the static analysis by Slither.

- False positives and these calls are safe in the context of the contracts

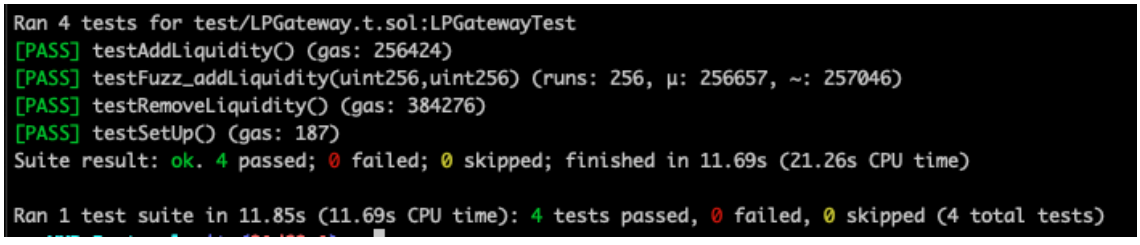


- Issues found NXDP-1



Unit Tests

All unit tests passed



However, we found that there is no test case for the *QDistributor* contract. It is noted that unit-testing for QDistributor is needed.

6. Unit-tests for QDistributor.

Issue ID

NXDP-6

Status

Resolved

#6382e5629263954b4be2c98abff3c9da525d783e

Risk Level

Severity: Medium

Description

Unit-testing is important for the *QDistributor* contract.

Recommendation

Add unit-tests for *QDistributor*

Conclusion

Arcadia identified issues that occurred at the following repository:

- **DXNhyperstructure/NXD-Protocol** at commit #91d60a134b6a8ca52e27356b70815e4c8c549dc2 as defined in the scope as in Section ‘Introduction and Audit Scope’
- A review of remediations for the issues at commit #6382e5629263954b4be2c98abff3c9da525d783e was also done.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, Arcadia and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.