Security Audit Report for DZapDiamondProxy Contracts

Date: October 23, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by BlockSec Audit Team
Date: 2023.10.23 11:38:47 +08'00'

Contents

Report Manifest

Item	Description
Client	DZapIO
Target	DZapDiamondProxy Contracts

Version History

Version	Date	Description
1.0	October 23, 2023	First Release

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repositiory of DZapDiamondProxy Contracts[1] of DZapIO. The DZapDiamondProxy Contracts serve as an aggregator for swaps and cross-chain transactions. Please be aware that, as stated by the developers, some of the smart contracts in this repository are forked from another repository (the link has been removed at the developers' request), which has previously been audited by third-party auditors. We thus assume these are considered reliable. Therefore, the focus of this part strictly lies in the differences between the forked code and its original source. Furthermore, the external dependencies, such as OpenZeppelin's library, are assumed to be reliable and are therefore not covered.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
DZapDiamondProxy Contracts	`Version 1`	`c88cfab094c06c3fd0efe3b4e515913483fb567e`
	`Version 2`	`7a6d7993e5faaa5fbf15f73e8447ade2fc8344b5`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the

[1] https://github.com/DZapIO/DZapCoreContracts

computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver

* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ **Likelihood**	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we do not find potential issues. Besides, we also have **four** recommendations and **three** notes.

- Recommendation: 4
- Note: 3

ID	Severity	Description	Category	Status
1	-	Remove duplicate checks	Recommendation	Fixed
2	-	Verify the new owner's address	Recommendation	Acknowledged
3	-	Optimize gas usage for the `ReentrancyGuard`	Recommendation	Fixed
4	-	Remove unused code	Recommendation	Fixed
5	-	Potential centralization risk	Note	-
6	-	Ensure no assets are left in the `DZapDiamond` contract	Note	-
7	-	Ensure the reliance and security of external protocols	Note	-

The details are provided in the following sections.

2.1 Additional Recommendation

2.1.1 Remove duplicate checks

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The check on Line 66 that verifies `msg.sender` is the contract owner is redundant, as this validation is already performed in the `onlyAuthorized` modifier.

```
65  function setFunctionApprovalBySignature(address _dex, bytes4 _signature, bool _approval)
        external onlyAuthorized {
66      if (msg.sender != LibDiamond.contractOwner()) LibAccess.enforceAccessControl();
67      if (_dex == address(this)) revert CannotAuthorizeSelf();
68
69      if (_approval) LibAllowList.addAllowedSelector(_dex, _signature);
70      else LibAllowList.removeAllowedSelector(_dex, _signature);
71
72      emit FunctionSignatureApprovalChanged(_dex, _signature, _approval);
73  }
```

Listing 2.1: DexManagerFacet.sol

```
15  modifier onlyAuthorized() {
16      if (msg.sender != LibDiamond.contractOwner()) LibAccess.enforceAccessControl();
17      _;
18  }
```

Listing 2.2: DexManagerFacet.sol

Impact N/A

Suggestion Remove duplicate checks accordingly.

2.1.2 Verify the new owner's address

Status Acknowledged

Introduced by Version 1

Description The transferOwnership function in the OwnershipFacet contract does not check that the _newOwner address is non-zero.

```
function transferOwnership(address _newOwner) external override {
    LibDiamond.enforceIsContractOwner();
    LibDiamond.setContractOwner(_newOwner);
}
```

Listing 2.3: OwnershipFacet.sol

Impact The contract owner may be incorrectly set to the invalid zero address.

Suggestion Add a check to avoid the zero address.

Feedback from the Project This is to revoke ownership and remove centralization in future when the platform becomes stable.

2.1.3 Optimize gas usage for the ReentrancyGuard

Status Fixed in Version 2

Introduced by Version 1

Description In the ReentrancyGuard contract, the _NOT_ENTERED and _ENTERED constants represent the status of the reentrant lock. For gas optimization, it is recommended to use values of 1 and 2 instead of 0 and 1. In Solidity, changing a storage variable from zero to non-zero costs more gas than changing between non-zero values.

```
uint256 private constant _NOT_ENTERED = 0;
uint256 private constant _ENTERED = 1;
```

Listing 2.4: ReentrancyGuard.sol

Impact N/A

Suggestion Change the value accordingly.

2.1.4 Remove unused code

Status Fixed in Version 2

Introduced by Version 1

Description In the LibSwap library, the invocation of the getRevertMsg function on Line 35 is unnecessary and can be removed.

```
    LibUtil.getRevertMsg(res);
```

Listing 2.5: LibSwap.sol

Impact N/A

Suggestion Remove the unused invocation accordingly.

2.2 Note

2.2.1 Potential centralization risk

Description The owner of the `DZapDiamond` contract has privileged capabilities such as adding/removing DEXes, modifying fees, and withdrawing all assets from the contract, posing concerns of potential centralization risks.

Feedback from the Project We plan to revoke ownership in future when the platform is stabilized.

2.2.2 Ensure no assets are left in the `DZapDiamond` contract

Description The protocol should prevent assets from being accidentally left in the `DZapDiamond` contract, as this opens up potential attack vectors. For example, in the `LibSwap` library, the `swap` function does not validate `msg.value` is greater than or equal to `fromAmount` when swapping native tokens. This lack of validation creates a potential vulnerability - if the contract holds native assets, an attacker could specify `fromAmount` as the contract's full balance while sending no native tokens. By exploiting the missing `msg.value` check, the attacker could drain native assets from the contract.

```
function swap(SwapData calldata _swapData, uint256 _totalTokenFees, bool _withoutRevert)
     internal returns (uint256 leftoverFromAmount, uint256 returnToAmount) {
    if (!LibAsset.isContract(_swapData.callTo)) revert InvalidContract();
    if (_swapData.fromAmount == 0) revert NoSwapFromZeroBalance();

    uint256 fromAmount = _swapData.fromAmount - _totalTokenFees;

    uint256 initialFromBalance = LibAsset.getOwnBalance(_swapData.from);
    uint256 initialToBalance = LibAsset.getOwnBalance(_swapData.to);
    uint256 nativeValue;

    if (LibAsset.isNativeToken(_swapData.from)) {
        nativeValue = fromAmount;
    } else {
        LibAsset.approveERC20(_swapData.from, _swapData.approveTo, fromAmount);
    }

    (bool success, bytes memory res) = _swapData.callTo.call{ value: nativeValue }(_swapData.
         swapCallData);

    if (!success) {
        if (_withoutRevert) {
            return (0, 0);
        }
```

```
            LibUtil.getRevertMsg(res);
            revert SwapCallFailed(res);
        }

        returnToAmount = LibAsset.getOwnBalance(_swapData.to) - initialToBalance;

        if (returnToAmount < _swapData.minToAmount) revert SlippageTooHigh(_swapData.minToAmount,
            returnToAmount);

        leftoverFromAmount = LibAsset.getOwnBalance(_swapData.from) - (initialFromBalance -
            fromAmount);
    }
```

Listing 2.6: LibSwap.sol

Feedback from the Project We are aware of the fact, but the one core idea of these contracts is that they won't hold any funds inside them, whatever the case may be.

2.2.3 Ensure the reliance and security of external protocols

Description As an aggregator protocol interacting with external DEXes and cross-chain protocols, it is crucial that the DZap protocol thoroughly validates any external protocols before integration. This validation process must thoroughly examine the reliance and security of external protocol code, as well as the correctness of interaction logic.

Feedback from the Project Yes before adding support for any dex we will validate it carefully. No dex will be added without rigorous testing to ensure that the security is not compromised.