Security Audit Report for DZap Aggregator Smart Contracts

Date: August 18, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.08.18 14:52:13 +08'00'

Contents

Report Manifest

Item	Description
Client	DZap.io
Target	DZap Aggregator Smart Contracts

Version History

Version	Date	Description
1.0	August 18, 2022	First Release

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

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 DZap Aggregator Smart Contracts [1]. The audit scope is limited to contracts under the `contracts` folder except the sub-folder `contracts/mock`. Specifically, the `DZapAggregator.sol` and `DZapDiscountNft.sol` are the key contracts of this project. The former is an aggregator platform that allows users to perform multiple kind of swaps among the native token (i.e., ETH and WETH), ERC tokens and LP tokens. The latter is designed to provide fee discounts for the swaps, i.e., some users may have fee discounts if they own the discount NFT tokens.

The auditing process is iterative. Specifically, we audit the initial version and following commits that fix the discovered issues. If there are new issues, we will continue this process. The commit hash 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
DZap Aggregator Smart Contracts	`Version 1`	`53f22702433f8e904556091aad305265176ac386`
	`Version 2`	`38820438659f735394b87edf5158d92e7a5dc175`

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 computing infrastructure are out of the scope.

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

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
* Access control
* 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 find **four** potential issues. We have **two** recommendations and **one** note.

- Low Risk: 4
- Recommendation: 2
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Unchecked function parameters	Software Security	Fixed
2	Low	Unverified `msg.value` and the corresponding parameters	DeFi Security	Fixed
3	Low	Lack of the balance check	DeFi Security	Fixed
4	Low	Controllable parameters for the `AGGREGATION_ROUTER.swap` function	DeFi Security	Fixed
5	-	Perform the early termination to save gas	Recommendation	Confirmed
6	-	Apply the `Pausable` mechanism for the emergency response	Recommendation	Confirmed
7	-	Potential risks of the controllable parameters	Note	Undetermined

The details are provided in the following sections. Note that, the code snippets related to the findings have been removed from the report upon the request of the developers.

2.1 Software Security

2.1.1 Unchecked function parameters

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the constructor of the `DZapAggregator` contract, some important parameters are not verified. Take the variable named `feeVault` as an example, which might be accidentally assigned as a zero address.

However, such a zero address may lead to problems. For example, at line 151 of the `swapTokensToTokens` function, `feeVault` will be used as the first argument to invoke the `_safenativeTransfer` function, and eventually it is used to trigger a native call. Note that the native call will always return TRUE even it is triggered by a zero address, and the require statement will be bypassed as well.

Besides, this variable is verified in the `updateFeeVault` function.

Impact May lead to unexpected results.

Suggestion Add sanity checks accordingly.

2.2 DeFi Security

2.2.1 Unverified `msg.value` and the correponding parameters

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `swapTokensToTokens` function, if `data.desc.srcToken` is native, the value to be transferred to invoke the Router's `swap` function is specified as `data.desc.amount` at line 138. However, `msg.value` passed into the `swapTokensToTokens` function is not verified. For the sake of simplicity, suppose `data_.length` is 1, which means the loop will be executed only once. If `msg.value` is not enough (`msg.value` < `data.desc.amount`), and the invocation of Router's `swap` function would fail. As such, at line 155, the caught exception might lead to transferring the specified amount back to the `msg.sender` from this contract (if this contract already has some native funds).

Impact The native funds of this contract might be harvested by the attacker.

Suggestion Add sanity checks accordingly.

2.2.2 Lack of the balance check

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `swapLpToTokens` function, the WETH balance will be calculated after swapping LP to WETH at line 191. However, the contract may has WETH before the swap (e.g., the existence of the `receiver` function would allow some accidentally transfers), which might be harvested by the attacker.

Impact The native funds of this contract might be harvested by the attacker.

Suggestion Revise the code accordingly.

2.2.3 Controllable parameters for the `AGGREGATION_ROUTER.swap` function

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description At line 138 of the `swapTokensToTokens` function, the `swap` function of `AGGREGATION_ROUTER` is invoked to swap tokens. However, all the parameters are attacker-controllable. Though the mock directory is NOT included in the audit, we took a look at the `AGGREGATION_ROUTER`'s implementation. We find that this function will invoke functions controlled by the attacker in the contract (specified by `data.executor` by the attacker). Since these parameters are controlled by the attacker, the implementation of the function can just return arbitrary values to the caller (without doing anything or just revert). In both cases, the `DZapAggregator` contract will transfer its tokens (if any) to the user specified by the attacker.

What's worse, the data.executor can be setup to the `DZapAggregator` contract itself. By doing so, the `data.executor.callBytes.selector` can be pointed to functions like `batchTransfer`. This can cause a reentrancy call to the batchTransfer function, which will transfer the tokens in the `DZapAggregator` contract (if any) to attacker-controlled addresses.

We highly recommend that there should be a whitelist of the data.executor and functions that can be invoked. Also, protection mechanisms like reentrancy guard would be helpful.

Impact The funds of the `DZapAggregator` contract might be harvested by the attacker.

Suggestion Revise the code accordingly.

Feedback from the Project Instead of whitelisting executor, we are now we have added a balance check for returnAmount.

2.3 Additional Recommendation

2.3.1 Perform the early termination to save gas

Status Confirmed

Introduced by `Version 1`

Description In the `DZapDiscountNft` contract, the require statement could be placed at the beginning of the `mintBatch` function (similar to the `mint` function) to perform the early termination to save gas.

Impact N/A

Suggestion Revise the code accordingly.

Feedback from the Project We don't want to add an extra for loop just for checking the ids.

2.3.2 Apply the `Pausable` mechanism for the emergency response

Status Confirmed

Introduced by `Version 1`

Description Applying the `Pausable` mechanism [1] is also recommended for security emergency purpose.

Impact N/A

Suggestion Apply this mechanism.

Feedback from the Project We can add it but the ideology using which we designed this contact was that every function will be atomic and the contact won't hold users tokens after the transaction is completed. So even if the contact gets compromised in any way we can just discard it as it won't be holding any funds.

2.4 Note

2.4.1 Potential risks of the controllable parameters

Status Undetermined

Introduced by `Version 1`

Description The functions of the `DZapAggregator` contract have a number of external controllable parameters, including the token addresses, token swapping paths and the caller to execute arbitrary functions. Currently such a design will NOT cause any critical issues due to the following two reasons:

[1] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/Pausable.sol

- this contract is merely served as a *proxy* to forward funds among users, the vault (i.e., feeVault) and the decentralized exchange (i.e., UniswapV2).
- this contract is not designed to keep funds.

However, it is worth noting that these parameters are manipulatable and it is necessary to be cautious of the potential risks.

Impact N/A

Suggestion Be cautious of the controllable parameters.