Security Audit Report for Duet Stablecoin Contracts

Date: Dec 07, 2021

Version: 1.0

Contact: contact@blocksecteam.com

Contents

Report Manifest

Item	Description
Client	DuetLab
Target	Duet Stablecoin Contracts

Version History

Version	Date	Description
1.0	Dec 07, 2021	First Release

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team 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 released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

The target contract is Duet Stablecoin Contracts. It is a platform that allows swapping between the platform's stablecoin (DUSD) and other stablecoins (including BUSD and USDC).

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

The MD5 values of the files before and after the audit are shown in the following:

Before

File	MD5
Contract Name	MD5
DuetUSDMinerPair.sol	a62fefde84ebe1e15a393fb9aa81b06c
DUSD.sol	8163bb4a82a4070e93af558df646913c
LinkUSDOracle.sol	400b5b556246fe1ce8267df42e99198d
chainlink/v0.6/interfaces/AggregatorV3Interface.sol	8e23314b120377fd91b8dca3a69acbf1
libs/TransferHelper.sol	ee431d96eaa23b41e6a72e9bbaf7099c
interfaces/IUSDOracle.sol	7ac09c9d5db25cbf99797cb63d24db37
interfaces/IDUSD.sol	7d9cb41fde64e1146e09a0b2a768ceea
interfaces/TokenRecipient.sol	6b12579c6158a0d302b150560be0752a

After

File	MD5
DuetUSDMinerPair.sol	a62fefde84ebe1e15a393fb9aa81b06c [1]
DUSD.sol	0febecb9e0095b7a2f3c6fd0c5f668dc [2]
LinkUSDOracle.sol	9657237e1d574cafc70d0cc70e42304a [3]
chainlink/v0.6/interfaces/AggregatorV3Interface.sol	8e23314b120377fd91b8dca3a69acbf1
libs/TransferHelper.sol	fc1283556b65e9dfd447155e2f44313b
interfaces/IUSDOracle.sol	7b8811f6af271ef7f0bdbf5668a68361
interfaces/IDUSD.sol	7d9cb41fde64e1146e09a0b2a768ceea
interfaces/TokenRecipient.sol	6b12579c6158a0d302b150560be0752a

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

[1] https://bscscan.com/address/0xe8521b61f64fc45a0bc3db36d2a524fe61a69b52#code and https://bscscan.com/address/0x7d3762b09d010957a551ad512fc0e53e6d3fb914#code

[2] https://bscscan.com/address/0xe04fe47516c4ebd56bc6291b15d46a47535e736b#code

[3] https://bscscan.com/address/0x22cd82e00d12315b440b88e518d035be62fb0e9d#code

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 do 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.

1.3 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 [4] and Common Weakness Enumeration [5]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

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

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

Chapter 2 Findings

In total, we find one potential issue in the smart contract, and we also have two recommendations, as follows:

- Medium Risk: 1
- Recommendation: 2

ID	Severity	Description	Category
1	Medium	*Miners Cannot Correctly Be Removed*	Software Security
2	-	*Do Not Use Elastic Supply Tokens*	Recommendation
3	-	*Potential Way to Optimize the* `bytesToUint` *Function*	Recommendation

The details are provided in the following sections.

2.1 Software Security

2.1.1 Miners Cannot Correctly Be Removed

Status Confirmed and fixed.

Description The flag used to indicate the removed miners in the `removeMiner` function is set to `true`, which means these miners cannot actually be removed.

```
28    function removeMiner(address _miner) public onlyOwner {
29        miners[_miner] = true;
30        emit MinerChanged(_miner, false);
31      }
```

Listing 2.1: removeMiner:DUSD.sol

Impact Once added, the miners cannot be removed anymore.

Suggestion Set the flag to `false`.

2.2 Additional Recommendation

2.2.1 Do Not Use Elastic Supply Tokens

Status Not an issue. The developers guarantee that the external stable coins are either USDC or BUSD.

Description & Suggestion Elastic supply tokens could dynamically adjust their price, supply, user's balance, etc. Such as inflationary token, deflationary token, rebasing token, and so forth. Such a mechanism makes a DeFi system over complex. For example, a DEX using deflationary token must double check the token transfer amount when taking swap action because of the difference of actual transfer amount and parameter. The abuse of elastic supply tokens will make the DeFi system vulnerable. In reality, many security accidents are caused by the elastic supply tokens. In terms of confidentiality, integrity and availability, we highly recommend that do not use elastic supply tokens.

Impact N/A

2.2.2 Potential Way to Optimize the `bytesToUint` Function

Status N/A

Description & Suggestion It is possible to use the *inline assembly code* [1] to optimize the `bytesToUint` function to save gas.

```
175    function bytesToUint(bytes calldata b) internal pure returns (uint256) {
176        uint256 number;
177        for(uint i=0; i < b.length; i++){
178            number = number + uint8(b[i])*(2**(8*(b.length-(i+1))));
179        }
180        return number;
181      }
```

Listing 2.2: bytestoUint:DuetUSDMinterPair.sol

Impact N/A

[1] https://github.com/GNSPS/solidity-bytes-utils/blob/master/contracts/BytesLib.sol

Chapter 3 Conclusion

In this audit, we have analyzed the business logic, the design, and the implementation of the Duet Stablecoin Contracts. Overall, the current code base is well structured and implemented, meanwhile, as previously disclaimed, this report does not give any warranties on discovering all security issues of the smart contracts. We appreciate any constructive feedback or suggestions.