BLOCKSEC

Security Audit Report for DefiBox Vault

Date: July 1, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	DefiBox
Target	DefiBox Vault

Version History

Version	Date	Description
1.0	July 1, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-07-02 09:37:15+0000

Chapter 1 Introduction

1.1 About Target Contracts

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

The focus of this audit is the DefiBox Vault. Specifically, the core contract (`vault.defi`) allows the users to deposit and redeem between EOS and SEOS. The deposited EOS will be reinvested to REX to generate profit for users.

It is important to note that only the C++ source files under the 'contracts' directory are included in the scope of this audit. Furthermore, all the dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and therefore, they are not included in the audit scope.

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.

Source	Version	File	MD5 Hash
Vault	Version 1	`stoken.defi/stoken.defi.cpp`	`35aedc2d7f1aa3d94f404eeb2af16a3d`
		`stoken.defi/stoken.defi.hpp`	`b669a9a15c6a223b46ebc5be57c19783`
		`stoken.defi/include/utils.hpp`	`a88cdc3685c8e1a59ad71b8f75e0b4d0`
		`vault.defi/vault.defi.cpp`	`033413e8bab74e9f1bd47d6da0b6b285`
		`vault.defi/vault.defi.hpp`	`0dc221ca11c6a148ed75f334a149d6cd`
		`vault.defi/src/tables.hpp`	`8af722e48ba740bf4346d90433260839`
		`include/defines.hpp`	`fc89af5e118b6f0b3f5e373ba78e3994`
	Version 2	`stoken.defi/stoken.defi.cpp`	`35aedc2d7f1aa3d94f404eeb2af16a3d`
		`stoken.defi/stoken.defi.hpp`	`b669a9a15c6a223b46ebc5be57c19783`
		`stoken.defi/include/utils.hpp`	`a88cdc3685c8e1a59ad71b8f75e0b4d0`
		`vault.defi/vault.defi.cpp`	`5afb79e093f6ac185ff166b00089b01b`
		`vault.defi/vault.defi.hpp`	`f301618e4ac991bcc56c4643eadf8703`
		`vault.defi/src/tables.hpp`	`8af722e48ba740bf4346d90433260839`
		`include/defines.hpp`	`dcbc2870a1850759b1dc096ad7dfe21a`

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 C++ language), the underlying compiling toolchain and the computing infrastructure (e.g., the blockchain runtime and system contracts of the EOS network) 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 [1] and Common Weakness Enumeration [2]. 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.

Table 1.1: Vulnerability Severity Classification

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

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

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

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.

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.

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **one** recommendation and **two** notes.

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

ID	Severity	Description	Category	Status
1	Medium	Potential underflow issue	Software Security	Fixed
2	Low	Duplicated issue symbols may point to the same collateral	Software Security	Acknowledged
3	-	Correct the typo	Recommendation	Fixed
4	-	Pontential centralization risks	Note	-
5	-	Potential stale REX price	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential underflow issue

Severity Medium

Status Fixed in Version 2

Introduced by Version 1

Description In the vault.defi contract, the check_for_released function is invoked when users withdraw EOS from the withdrawal queue. The EOS amount is calculated by multiplying the REX amount at the time of release by the most recent REX price and then minus the banks.size(), as shown in the following code snippet.

```
uint64_t refund_amount = uint128_t(withdraw.rex) * rex_price / RATE_BASE - withdraw.banks.size
    ();
```

Listing 2.1: contracts/vault.defi/vault.defi.cpp

However, there is a potential underflow in this calculation. Specifically, a malicious user can manipulate withdraw.rex and withdraw.banks.size() to make the uint128_t(withdraw.rex) * rex_price / RATE_BASE equal to 0 due to precision loss. As a result, the subtraction operation will underflow and cause unexpected behaviors.

Impact Potential underflow may bring unexpected behaviors.

Suggestion Add underflow checks.

2.1.2 Duplicated issue symbols may point to the same collateral

Severity Low

Status Acknowledged

Introduced by `Version 1`

Description In the `vault.defi` contract, the `do_release` function accepts different wrapped tokens (S-Tokens) from users and queues corresponding withdrawals. Currently, the contract only supports the S-Tokens of SEOS. However, different tokens may share the same symbol, as the symbols of the S-Tokens are derived solely from the symbol of the collateral token, regardless of the token contract. As a result, S-Tokens with different collateral tokens may also share the same symbol.

When the `do_release` function is invoked, the contract searches for the collateral token according to the symbol of the input S-Token. If there are multiple collateral tokens that share the same symbol, the search procedure may not work as expected.

```
void vault::do_release(const name &code, const name &owner, const asset &quantity, const string
     &memo) {
// check
check(_config.withdraw_status == 1, "withdraw has been suspended");
check(quantity.symbol == SEOS_SYMBOL, "only support SEOS");

auto bank = parse_name(memo);
check(banks.find(bank) != banks.end(), "Please fill in the correct account in the memo.");

// save daily rex price
save_daily_rex_price();

// withdraw matured rex
withdraw_sell_rex(bank);

// mvfrsavings
auto collateral = get_collateral_by_issue_symbol(quantity.symbol);
```

Listing 2.2: contracts/vault.defi/vault.defi.cpp

Impact Duplicate issue symbols may point to the same collateral.

Suggestion Refactor the logic accordingly.

Feedback from the Project The project will only add EOS and USDT as the collateral tokens, and there will be no support of other collateral tokens in the future.

2.2 Additional Recommendation

2.2.1 Correct the typo

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Several typos need to be corrected in the project. Specifically, all instances of `STOKRN_ACCOUNT` should be corrected to `STOKEN_ACCOUNT`.

Suggestion Correct the typo.

2.3 Note

2.3.1 Pontential centralization risks

Description The project introduces several mechanisms that can increase the risk of centralization, including the ability to modify critical configurations and manage leftover EOS in the vault contract.

2.3.2 Potential stale REX price

Description The `check_for_released` function queries the previously saved daily REX prices to find the most recent price. In extreme cases, there may be one or more days without prices recorded due to inactive interactions with the protocol. As a result, the fetched price may be stale, causing users to receive less EOS than expected.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS