Sugar Bounce

Smart Contract Security Audit

V1.0

No. 202204121132

Apr 12th, 2022

Contents

Summary of audit results 1

1 Overview 2

1.1 Project Overview 2

1.2 Audit Overview 2

2 Findings 3

[SB-1] Signature reuse 4

3 Appendix 5

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 5

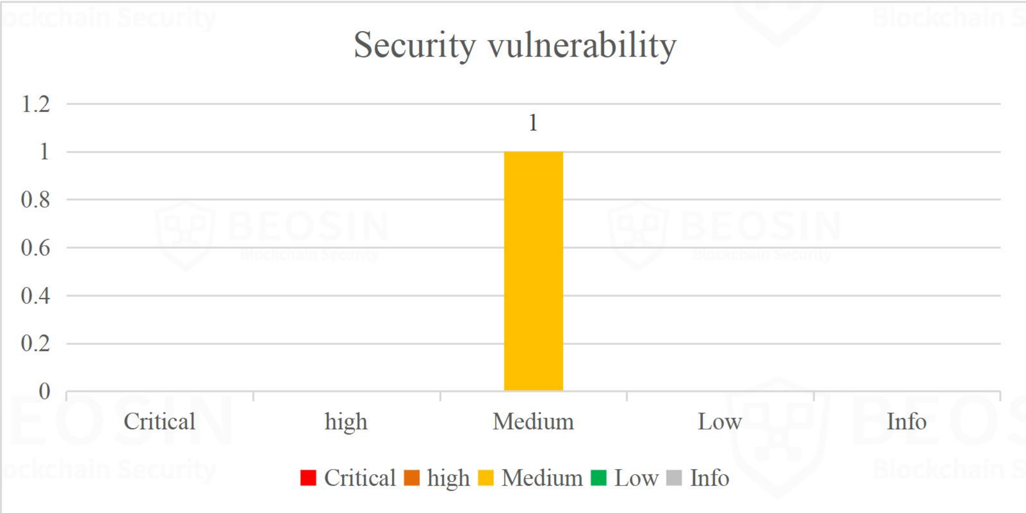
3.2 Audit Categories 7

3.3 Disclaimer 9

3.4 About BEOSIN 10

Summary of audit results

After auditing, 1 Medium-risk item was identified in the Sugar Bounce project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



*Notes:

- **Risk Description:**

1. This project has an off-chain signature authentication function. The off-chain signature security is not included in the scope of audit.

- **Project Description:**

1. Business overview

The project has two parts, *permitSBToken* and *buyCredit*, which perform approve and transfer functions respectively, both of which are off-chain signature authentication.

1 Overview

1.1 Project Overview

Project Name	Sugar Bounce
Platform	BNB Chain
Github Link	https://github.com/SugarBounceNSFW/sugrabounce-buycredit-smartcontract
Commits	c269348de574d9ed285aca500f1986235ca87e5e(origin) 6135a39aad6122909a1ac0a6e545d74fd7cf650e(fixed)

1.2 Audit Overview

Audit work duration：April 11, 2022 – April 12, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Technology Co. Ltd.

2 Findings

Index	Risk description	Severity level	Status
SB-1	Signature reuse	Medium	Fixed

*SB is short for Sugar Bounce project

[SB-1] Signature reuse

Severity Level Medium

Type Business Security

Lines BuyCredit.sol#L48

Description In the BuyCredit.sol contract, the buyCredit function does not add nonce when verifying the signature hash. If the from address has sufficient authorization for the BuyCredit contract, the user can reuse the same signature to call the buyCredit function to obtain additional tokens.

```
37    function buyCredit(
38        address from,
39        address to,
40        uint256 amount,
41        uint256 deadline,
42        uint8 v,
43        bytes32 r,
44        bytes32 s
45    ) external nonReentrant {
46        require(deadline > block.timestamp, "Transfer signature is expired");
47
48        bytes32 msgHash = keccak256(abi.encode(_BUY_CREDIT_TYPEHASH, from, to, amount, deadline));
49
50        // bytes32 msgHash = keccak256(abi.encodePacked(productName));
51        bytes32 digest = toTypedMessageHash(msgHash);
52        // (bytes32 r, bytes32 s, uint8 v) = splitSignature(sig);
53        address recoveredAddress = ecrecover(digest, v, r, s);
54
55        require(recoveredAddress == from, "Invalid signer");
56
57        TransferHelper.safeTransferFrom(SB_TOKEN, from, to, amount);
58
59        emit TransferSB(from, to, amount);
60    }
61 }
```

Figure 1 Source code of *buyCredit* function(Unfixed)

Recommendations It is recommended that the signature data should contain a nonce.

Status **Fixed.**

```
43    function buyCredit(
44        address from,
45        address to,
46        uint256 amount,
47        uint256 deadline,
48        uint8 v,
49        bytes32 r,
50        bytes32 s
51    ) external nonReentrant {
52        require(deadline > block.timestamp, "Transfer signature is expired");
53
54        bytes32 msgHash = keccak256(abi.encode(_BUY_CREDIT_TYPEHASH, from, to, amount, deadline, nonces[from]));
55
56        // bytes32 msgHash = keccak256(abi.encodePacked(productName));
57        bytes32 digest = toTypedMessageHash(msgHash);
58        // (bytes32 r, bytes32 s, uint8 v) = splitSignature(sig);
59        address recoveredAddress = ecrecover(digest, v, r, s);
60
61        require(recoveredAddress == from, "Invalid signer dfgdgdfgdf");
62
63        TransferHelper.safeTransferFrom(SB_TOKEN, from, to, amount);
64
65        nonces[from]++;
66
67        emit TransferSB(from, to, amount);
68    }
69 }
```

Figure 2 Source code of *buyCredit* function(Fixed)

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party protocol interface consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable token price
		Centralized asset control
		Asset tradability
		Arbitrage attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The peoject party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

Affiliated to BEOSIN Technology Pte. Ltd., BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com