Code Security Assessment

BUTTER NETWORK

August 1st, 2024

Contents

CONTENTS....1
SUMMARY....2
ISSUE CATEGORIES....3
OVERVIEW....4
PROJECT SUMMARY....4
VULNERABILITY SUMMARY....4
AUDIT SCOPE....5
FINDINGS....6
INFORMATIONAL....7
EKD-01 | Use "End...Parse()"....**7**
DESCRIPTION....7
RECOMMENDATION....8
INFORMATIONAL....9
EKD-02 | Use Nested Storage....9
DESCRIPTION....9
RECOMMENDATION....10
DISCLAIMER....11
APPENDIX....12
ABOUT....13

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best prac tices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	BUTTER NETWORK
Platform	TON
Website	https://butternetwork.io
Type	Bridge
Language	Func
Codebase	https://github.com/butternetwork/ton-router-contracts

Vulnerability Summary

Vulnerability Level	Total	Mitigated	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	0	0	0	0	0	0
Informational	2	0	0	0	0	2
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
EKD	ton_router.fc	A665A45920422F9D417E4867EFDC4FB8A04A1F3FFF1FA07E998E86F7F7A27AE3

Findings

ID	Issue	Severity	Status
EKD-01	Use "end_parse()"	Informational	Resolved
EKD-02	Gas Optimization	Informational	Resolved

INFORMATIONAL

EKD-01 | Use "End...Parse()"

Issue	Severity	Location	Status
Use `end_parse()`	Informational	ton_router.fc	Resolved

Description

```
28 () load_data() impure {
        var ds = get_data().begin_parse();

        ctx_order_id = ds~load_uint(64);
        ctx_owner = ds~load_msg_addr();

        var addresses_slice = ds~load_ref().begin_parse();
        ctx_withdrawer = addresses_slice~load_msg_addr();
        ctx_bridger = addresses_slice~load_msg_addr();
        ctx_bridge_token_address = addresses_slice~load_msg_addr();

    ds.end_parse();
}
```

When reading data from storage, try to use end_parse(). Maintaining this habit helps ensure the consistency of data flow between writing and reading, reducing unexpected errors.

Recommendation

It is recommended to add `addresses_slice.end_parse();`.

INFORMATIONAL

EKD-02|Use Nested Storage

Issue	Severity	Location	Status
Gas Optimization	INFORMATIONAL	ton_router.fc	Resolved

Description

```
13 ;; Order ID, used to track bridge operations (ton -> other chain)
global int ctx_order_id;

    ;; Contract owner address
global slice ctx_owner;

    ;; Authorized address for withdrawal operations
global slice ctx_withdrawer;

    ;; Authorized address for bridge in operations (other chain -> ton)
global slice ctx_bridger;

    ;; Bridge token address, usdt jetton wallet address of this contract
global slice ctx_bridge_token_address;
```

Design some data that needs to be stored in contract storage in a nested manner. Only unpack it when needed, as this helps save gas.

Recommendation

It is recommended to redesign the `load_data` and `save_data` methods using the following pattern. However, this modification is not mandatory; using this pattern will require a significant amount of code to be redesigned.

```
(int, int, slice, slice) load_data() inline {
    slice ds = get_data().begin_parse();
    var data = (
        ds~load_uint(STATUS_SIZE), ;; status
        ds~load_coins(), ;; balance
        ds~load_msg_addr(), ;; owner_address
        ds~load_msg_addr() ;; jetton_master_address
    );
    ds.end_parse();
    return data;
}

() save_data(int status, int balance, slice owner_address, slice
jetton_master_address) impure inline {
    set_data(pack_jetton_wallet_data(status, balance, owner_address,
jetton_master_address));
}
```

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

- Ethereum
- Cosmos
- Eos
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- c++

CONTACTS

- https://dehacker.io
- https://twitter.com/dehackerio
- https://github.com/dehacker/audits_public
- https://t.me/dehackerio
- https://blog.dehacker.io/

DeHacker

August 1st 2024