GUILD AUDITS

Chain Thrift

Initial Audit Report

October 10th, 2022

Contents

Executive Summary

This section will represent the summary of the whole audit process once it has concluded.

Project Summary

Project Name: Chain Thrift

Description: ChainedThrift is a decentralized finance application that is built on blockchain technology. It is built on the Polygon (Matic) blockchain network. Chained Thrift helps its users achieve their financial goals through its decentralized thrift saving scheme. It also provides autonomous trading options on its decentralized exchange.

Codebase: github

Commit: initial-audit c926c4819d7bd5669a93cf47f439386a6579d383

Project Audit Scope and Findings

The motive of this audit is to review the codebase of Chain Thrift contracts for the purpose of achieving secured, corrected and quality contracts.

Number of Contracts in Scope:
- purse (540 SLOC)
- purseFactory (83 SLOC)

Duration for Audit: October 03, 2022 to October 17, 2021

Mode of Audit and Methodologies

The mode of audit carried out in this audit process is as follow:

- **Manual Review**: This is the first and principal step carried out to understand the business logic behind a project. At this point, it involves research, discussion and establishment of familiarity with contracts. Manual review is critical to understand the nitty-gritty of the contracts.
- **Automated Testing**: This is the running of tests with audit tools to further cement discoveries from the manual review. After a manual review of a contract, the audit tools which could be Slither, Echidna, or Mythril are run on the contract to find out issues.
- **Functional Testing**: Functional testing involves manually running unit, static, and dynamic testing. This is done to find out possible exploit scenarios that could be used to steal money from the contracts. This helps understand the functionality of the contracts and find out lapses in the reverts check in contract.

The methodologies for establishing severity issues:

- High Level Severity Issues
- Medium Level Severity Issues
- Low Level Severity Issues
- Informational Level Severity Issues

Report of Findings

HIGH SEVERITY ISSUES

Issue: Array length not validated

Severity: High

Context: purse.sol L455 L427

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L455

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L427

Description:

The length of the memory array used to store the array of member who did not donate, will not catch the total members in the purse because the logic (“ i < array.length-1”) does not catch the full length of the members of the array of the purse because it is reduced by 1.

Impact of the vulnerability

The impact of the error will prompt the function to miss out from catching the last member of the array with an error message “Index out of bound” therefore the last member of the array won’t be able to get their collateral and interest.

Recommendation:

It is recommended that the “-1” in the for loop should be removed.

Issue: Logical error

Severity: High

Context: purse.sol L469

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L469

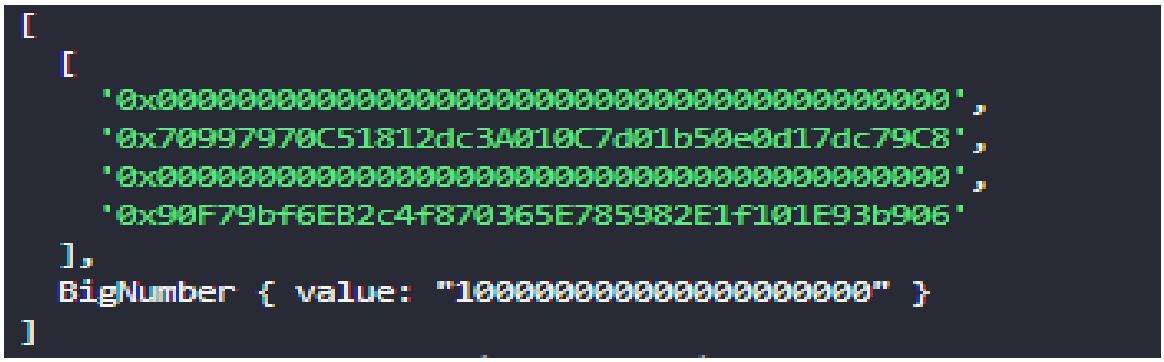
Description:

The function **calculateMissedDonationByUser** was supposed to return an array of members that the '_memberAdress' do not donate to but because the array of members was loop using a fixed array in memory, this will initially populate the resulting array with zero address before pushing the member array. This cause wrong calculation of funds on
`members_who_member_didnt_donate_for.length * purse.deposit_amount`

The resulting amount by this user will be multiplied by the original member they missed * the remaining zero address in the array

Impact of the vulnerability

Assuming a user missed two donations, the resulting array will be the two address they missed plus the initial zero address. See image below.



With the deposit amount of 25 ethers, the total amount of donations missed by the user is 25 * the array.length = 100 ether instead of 50 ethers.

Recommendation:

A local variable of uint256 num, should be created, the count will only be increased when the check is passed. See image below

```
        uint256 num;

        for (uint256 i = 0; i < members.length; i++) {
            if (
                members[i] != _memberAdress &&
                has_donated_for_member[_memberAdress][members[i]] == false
            ) {

                members_who_member_didnt_donate_for[i] = (members[i]);
                num++;
            }
        }

        return (
            members_who_member_didnt_donate_for,
             num * purse.deposit_amount
        );
```

Issue: Logical error

Severity: High

Context: purse.sol L420

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L420

Description:

The function **calculateMissedDonationForUser** was supposed to return an array of members that do not donate for '_memberAdress' but because the array of members was loop using a fixed array in memory, this will initially populate the resulting array with zero address before pushing the member array. This cause wrong calculation of funds on `members_who_didnt_donate_for_user.length * purse.deposit_amount`

The resulting amount for this user will be multiplied by the original member that didn't donate * the remaining zero address in the array

Impact of the vulnerability

Assuming no donation was missed for this particular user, the returned array will be filled with zero address in length of the total user in the thrift multiplied by the total amount. On withdrawing collateral and yield, the thrift will pay the user the deposit amount * all the total number of thrift members

Recommendation:

A local variable of uint256 num, should be created, the count will only be increased when the check is passed. See image below

```
        uint256 num;

        for (uint256 i = 0; i < members.length; i++) {
            if (
                members[i] != _memberAdress &&
                has_donated_for_member[members[i]][_memberAdress] == false
            ) {
                members_who_didnt_donate_for_user[i] = (members[i]);
                num++;
            }
        }

        return (
            members_who_didnt_donate_for_user,
            num * purse.deposit_amount
        );
```

Issue: Unchecked transfer

Severity: High

Context: purse.sol L233-237, L276-280, L310, L381-415, L498

Description:

The return value of an external transfer/transferFrom call is not checked

Impact of the vulnerability

Several tokens do not revert in case of failure and return false. If one of these tokens is used in purse, deposit will not revert if the transfer fails, and an attacker can call claim, withdraw or deposit for free

Recommendation:

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

MEDIUM SEVERITY ISSUES

Issue: Inability to get back collateral when the required number of members to start a purse is not met.

Severity: Medium

Context:

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol

Description:

In the contract, there is no provision for the purse members to be claim there money back when the duration of the purse last for a long time without the required amount of users joining the purse before it kicks off.

Impact of the vulnerability

The money will be trapped in the contract as far as the required numbers of members has not joined the purse.

Recommendation:

Members should be able to have options of getting the funds back when it takes a very long time to kick off.

Issue: Contract that lock Ether

Severity: Medium

Context:

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L171

Description:

The purse.sol constructor is make as payable which means ether can be sent into the contract but there is no function handling withdrawal capacity for ether

Impact of the vulnerability

Every Ether sent into purse contract will be lost.

Recommendation:

Remove the payable attribute in the constructor as the purse is only using erc20 for transcations.

LOW SEVERITY ISSUES

Issue: Looping storage variables

Severity: Low

Context:

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L229>

Description:

Looping variable in the storage cost gas consumption because every read to the storage consumes a high gas cost and looping should be avoided if possible to avoid users paying high gas cost for performing a transaction.

Recommendation:

If looping will be used in a function it is recommended to loop through a memory variable which will cost less gas: A memory variable should be declare and assign the state variable because looping through the memory variable cost less gas.
The code snippet below is suggested.

INFORMATIONAL SEVERITY ISSUES

Issue: Inefficient storage layout

Severity: Informational

Context: purse.sol L58

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L58

Description:

The **struct purse**' S declaration is inefficient and causes a stack too deep error as the struct's properties can be break into another struct.

Recommendation:

We recommend to remove **uint256 purseId** has the variable is currently unused and taking a whole slot in storage.

Issue: State variable that could be declared constant

Severity: Informational

Context: purse.sol L123

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purse.sol#L123

Description:

The bentoBox_address state variables should be declared constant to save gas as the address will remain constant.

Recommendation:

Add the constant attributes to state variables that never change.

Issue: write after write

Severity: Informational

Context: purseFactory.sol L64

https://github.com/Bloceducare/chainedThrift_contracts/blob/c926c4819d7bd5669a93cf47f439386a6579d383/contracts/purseFactory.sol#L64

Description:

```
purse_count = purse_count++;
```

here purse_count variable was written but never read and written again

Recommendation:

Change the line of code to `purse_count++`

Closing Summary

There were discoveries of some high, medium, low and informational issues after the audit. The audit team thereafter suggested some remediation to help remedy the issues found in the contract.

Appendix

- Audit: The review and testing of contracts to find bugs.
- Issues: These are possible bugs that could lead exploits or help redefine the contracts better.
- Slither: A tool used to automatically find bugs in a contract.
- Severity: This explains the status of a bug.

Disclaimer

While the audit report is aimed at achieving a quality codebase with assured security and correctness, it should not be interpreted as a guide or recommendation for people to invest in **chain thrift** contracts.

With smart contract audit being a multifaceted process, we admonish the **chain thrift** team to carry out further audit from other audit firms or provide a bug bounty program to ensure that more critical audit is done to the contract.

Guild Audits

Guild Audits is geared towards providing blockchain and smart contract security in the fuming web3 world. The firm is passionate about remedying the constant hacks and exploits that deters the web3 motive.