decurity

Smart Contract Security Audit Report

Chainspot

1. Contents

2. General Information

This report contains information about the results of the security audit of the Chainspot (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 10/06/2024 to 13/06/2024.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/Chainspot-router/. Initial review was done for the commit 9caa1c5 and the re-testing was done for the commit 1eeb48.

The following contracts have been tested:

- contracts/base/LoyaltyNFTV1.sol
- contracts/utils/AddressLib.sol
- contracts/utils/CountersLib.sol
- contracts/utils/SafeMath.sol
- contracts/ChainspotProxyV1.sol
- contracts/LoyaltyCashbackV1.sol
- contracts/LoyaltyNFTClaimerV1.sol
- contracts/LoyaltyReferralV1.sol
- contracts/ProxyFee.sol
- contracts/ProxyWithdrawal.sol

2.3. Threat Model

The assessment presumes the actions of an intruder who might have the capabilities of any role (an external user, token owner, token service owner, or a contract).

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a reentrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, and the impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises the best effort to perform its contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using limited resources.

3. Summary

As a result of this work, we have discovered medium level security issues, which has been fixed and re-tested in the course of the work.

As a result of this work, we have discovered a several medium security issues. The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement). These vulnerabilities have been addressed and thoroughly re-tested as part of our process.

The Chainspot team has given feedback for the suggested changes and an explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of October 31, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Logic error in `transferBaseFee`	contracts/ChainspotProxyV1.sol	**Medium**	Fixed
No storage gaps for upgradeable contracts	contracts/ProxyFee.sol	**Medium**	Fixed
Centralizations risks	contracts/ChainspotProxyV1.sol	**Medium**	Acknowledged
Base fee doesn’t have limitations	contracts/ProxyFee.sol	**Medium**	Acknowledged
SafeERC20 library is imported but never used	contracts/ProxyWithdrawal.sol contracts/ChainspotProxyV1.sol	**Medium**	Fixed
Revert on Zero Value Approvals	contracts/ChainspotProxyV1.sol	**Medium**	Fixed
Lack of user and ref level check	contracts/ChainspotProxyV1.sol	**Low**	Acknowledged

Issue	Contract	Risk Level	Status
Use two step ownership transfer	contracts/base/LoyaltyNFTV1.sol contracts/ProxyWithdrawal.sol contracts/ProxyFee.sol	Low	Fixed
Fee payment can be bypassed	contracts/ChainspotProxyV1.sol	Low	Not fixed
Fee payment can be bypassed	contracts/ChainspotProxyV1.sol	Low	Acknowledged

4. General Recommendations

This section contains general recommendations on how to improve the overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Logic error in `transferBaseFee`

Risk Level: Medium

Status: FIxed in the commit 9f2c5a0a.

Contracts:

- contracts/ChainspotProxyV1.sol

Location: Function: `transferBaseFee`.

Description:

There is a `transferBaseFee()` that is responsible for distributing fees across owner and referral:

```
contracts/ChainspotProxyV1.sol:
    function transferBaseFee(uint _amount, uint8 _userLevel, address _referrer, uint8 _refLevel, bool _isNativeTransfer) private returns(uint) {
        uint baseFeeAmount = calcBaseFee();
        if (baseFeeAmount == 0) {
            return 0;
        }
        uint additionalFee = _isNativeTransfer ? calcAdditionalFee(_amount) : 0;

        uint finalBaseFeeAmount = baseFeeAmount;
        if (_userLevel > 0 && _referrer != address(0) && _refLevel > 0) {
                                            ...
                uint refAmount = baseFeeAmount.mul(refererLevelData.refProfitInPercent).div(100);
                if (refAmount > 0) {
                    finalBaseFeeAmount = finalBaseFeeAmount.sub(refAmount);
                    referral.addRefererProfit{value: refAmount}(_referrer);
                }
            }
        }

        (bool successTV, ) = owner().call{value: finalBaseFeeAmount + additionalFee}("");
        require(successTV, "ChainspotProxy: fee not sent");

```

```
  206:         return _isNativeTransfer ? additionalFee : baseFeeAmount +
additionalFee; //@audit logic error
  207:     }
```

In case it's a native transfer a function is taking an additional fee and sending it to the owner on the 203 line. That means that the amount that is equal to the `msg.value` in this call was reduced by `baseFeeAmount + additionalFee` . But as we can see a `transferBaseFee()` is returning only `additionalFee`.

This flow may bypass this check:

```
contracts/ChainspotProxyV1.sol:
  124:         uint amountWithoutFee = amount.sub(transferBaseFee(_amount,
_userLevel, _referrer, _refLevel, true));
  125:         require(amountWithoutFee >= _targetAmount, "ChainspotProxy:
routerAmount is too small");
```

The `transferBaseFee` was subtracted from the `amount` but actually, we didn't subtract `baseFeeAmount`.

This could lead to sweeping native tokens from this contract that was left there and should be able to sweep only by the owner through `transferCoins()` or failing of call because of not enough funds.

Remediation:

Consider changing the return value to:

```
return baseFeeAmount + additionalFee;
```

5.2. No storage gaps for upgradeable contracts

Risk Level: Medium

Status: Fixed in the commit 9f2c5a0a.

Contracts:

- contracts/ProxyFee.sol

Description:

For upgradeable contracts, there must be a storage gap to allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployment. Otherwise, it may be very difficult to write new implementation code.

Without a storage gap, the variable in the child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences to the child contracts, potentially causing loss of user funds or **causing** the contract to malfunction completely.

Remediation:

Consider adding an appropriate storage gap at the end of the ProxyFee contract such as the below:

```
uint256[50] private __gap;
```

5.3. Centralizations risks

Risk Level: Medium

Status: Acknowledged

Contracts:

- contracts/ChainspotProxyV1.sol

Description:

If the owner adds the token contract to the clients, it will be possible to steal all tokens approved by users.

```
// @audit centralization risks
function addClient(address _clientAddress) public onlyOwner {
    require(_clientAddress.isContract(), "ChainspotProxy: address is non-
contract");
    clients[_clientAddress].exists = true;
    emit AddClientEvent(_clientAddress);
}
```

The owner or anyone else can steal the user’s tokens by forming certain** **data.

```
function proxyCoins(
    address _to, uint _amount, uint _targetAmount, uint8 _userLevel,
    address _referrer, uint8 _refLevel, bytes calldata _data
) internal {
    ...
    (bool success, ) = _to.call{value: _targetAmount}(_data);
    require(success, "ChainspotProxy: transfer not sent");
    }
```

Because the approval of the tokens is given to the ChainSpotProxtV1 contract and this contract is upgradeable it also creates a risk of stealing the funds.

Remediation:

Consider transferring the ownership to the Timelock which will add an extra layer of security by delaying any changes that could potentially be malicious or harmful. This will give users more time to react if anything suspicious happens.

5.4. Base fee doesn't have limitations

Risk Level: Medium

Status: Acknowledged

Contracts:

- contracts/ProxyFee.sol

Description:

The owner can charge an arbitrary commission for transfers with tokens.

```
// @audit-issue the owner can set an arbitrary rate and increase the fee to
any values
function updateRate(uint _rate) external onlyOwner {
    rate = _rate;
}

function calcBaseFee() internal view returns(uint) {
    return baseFeeInUsd * rate;
}
```

Remediation:

Add a temporary lock when setting a new commission to inform users.

5.5. SafeERC20 library is imported but never used

Risk Level: Medium

Status: Fixed in the commit 9f2c5a0a.

Contracts:

- contracts/ProxyWithdrawal.sol

- contracts/ChainspotProxyV1.sol

Description:

There are IERC20 functions that are called directly in the protocol:

```
contracts/ProxyWithdrawal.sol:
  47:          require(_token.transfer(_to, _amount), "Withdrawal: transfer
request failed");

contracts/ChainspotProxyV1.sol:
  154:                  require(_token.transferFrom(msg.sender, owner(),
feeAmount), "ChainspotProxy: fee transfer request failed");
  159:              require(_token.transferFrom(msg.sender, address(this),
routerAmount), "ChainspotProxy: transferFrom request failed");
  161:              require(_token.approve(_approveTo, routerAmount),
"ChainspotProxy: approve request failed");
  168:              require(_token.approve(_approveTo, 0), "ChainspotProxy:
revert approve request failed");
```

Since the IERC20 interface requires a boolean return value, attempting to `transfer()`, `approve()` , and `transferFrom()` ERC20s with missing return values will revert. This means Chainspot won't support several popular ERC20s, including USDT and BNB tokens on the Ethereum chain.

Remediation:

Consider using a safe version of the mentioned functions from SafeERC20 library such as `safeTransfer()`, `safeTransferFrom()`, `forceApprove()` that handles a case when functions may not return values.

References:

- https://github.com/d-xo/weird-erc20#missing-return-values

5.6. Revert on Zero Value Approvals

Risk Level: Medium

Status: Fixed in the commit 9f2c5a0a.

Contracts:

- contracts/ChainspotProxyV1.sol

Description:

Some tokens (e.g. BNB) revert when approving a zero value amount (i.e. a call to `approve(address, 0)`).

```
contracts/ChainspotProxyV1.sol:
  167:         if (_token.allowance(address(this), _approveTo) > 0) {
  168:             require(_token.approve(_approveTo, 0), "ChainspotProxy: revert approve request failed"); //@audit won't work for BNB token for ex.
  169:         }
```

Remediation:

Consider refactoring the logic to catch this and make the protocol compatible with the Ethereum version of the BNB token.

For example with wrapping approve call into `try/catch`.

5.7. Lack of user and ref level check

Risk Level: Low

Status: Acknowledged

Contracts:

- contracts/ChainspotProxyV1.sol

Location: Function: `transferBaseFee`.

Description:

There is a `metaProxy()` that accepts `_userLevel` and `_refLevel` parameters responsible for the percentage that will have to be transferred to the referral in `transferBaseFee()`.

Unfortunately, there is no check during the call that verifies that passed levels are correct which creates a possibility of the manipulation for the caller of this function.

Remediation:

Consider verifying that `_userLevel` and `_refLevel` values are correct.

5.8. Use two step ownership transfer

Risk Level: Low

Status: Fixed in the commits 9f2c5a0a and 1eeb48.

Contracts:

- contracts/base/LoyaltyNFTV1.sol
- contracts/ProxyWithdrawal.sol
- contracts/ProxyFee.sol

Description:

The contract uses OZ `OwnableUpgradeable`. Abstract contract `OwnableUpgradeablea` has a `transferOwnership()`. But the issue here is that if the owner called `transferOwnership()` and passed the wrong address, the ownership over a contract will be lost. So before transferring ownership, it is better to have some sort of verification, that there is control over a new owner's address.

Remediation:

Consider using `Ownable2StepUpgradeable` which already implemented the described logic which prevents the contract ownership from accidentally being transferred to an address that cannot handle it.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol

5.9. Fee payment can be bypassed

Risk Level: Low

Status: Acknowledged

Contracts:

- contracts/ChainspotProxyV1.sol

Description:

In cases where the user needs to transport a large amount of ETH through the protocol, he can bypass the payment of the native transfer fee using the token transfer function.

```
function proxyTokens(
    IERC20 _token, uint _amount, uint _targetAmount, address _approveTo,
address _callDataTo,
    uint8 _userLevel, address _referrer, uint8 _refLevel, bytes calldata _data
) internal {
    ...
    // @audit-issue the msg.value will still be transferred in full, except
for a lower commission
    (bool success, ) = _callDataTo.call{value:
msg.value.sub(transferBaseFee(_amount, _userLevel, _referrer, _refLevel,
false))}(_data);
    require(success, "ChainspotProxy: call data request failed");
    ...
}
```

The second way to bypass the paying fee is to send _amount equal to zero in the `proxyCoins()` call.

```
contracts/ChainspotProxyV1.sol:
          function proxyCoins(
              address _to, uint _amount, uint _targetAmount, uint8
_userLevel,
              address _referrer, uint8 _refLevel, bytes calldata _data
          ) internal {
              uint amount = msg.value;
              require(amount > 0, "ChainspotProxy: zero amount");
              require(amount >= _amount, "ChainspotProxy: amount is too
small");

              uint amountWithoutFee = amount.sub(transferBaseFee(_amount,
_userLevel, _referrer, _refLevel, true));
              require(amountWithoutFee >= _targetAmount, "ChainspotProxy:
routerAmount is too small");

              (bool success, ) = _to.call{value: _targetAmount}(_data);
              require(success, "ChainspotProxy: transfer not sent");
          }
```

The fee is calculated based on `_amount` value and the function only checks that this value is not bigger than the `msg.value` so we can set it as zero and pay only `baseFee`.

: Remediation

When calculating the commission for the transfer of tokens, take a fee from both tokens and ETH or restrict sending `msg.value` through `proxyTokens()`.

To fix the second bypass consider calculating the fee based on the passed `msg.value` instead of `_amount`.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.