MONETHIC

BJustCoin

Smart Contract Audit Report

Prepared for:
BJustCoin s.r.o.

Version:
Final, 20.12.2024

Table of Contents

About Monethic

Monethic is a young and thriving cybersecurity company with extensive experience in various fields, including Smart Contracts, Blockchain protocols (layer 0/1/2), wallets and off-chain components audits, as well as traditional security research, starting from penetration testing services, ending at Red Team campaigns. Our team of cybersecurity experts includes experienced blockchain auditors, penetration testers, and security researchers with a deep understanding of the security risks and challenges in the rapidly evolving IT landscape. We work with a wide range of clients, including fintechs, blockchain startups, decentralized finance (DeFi) platforms, and established enterprises, to provide comprehensive security assessments that help mitigate the risks of cyberattacks, data breaches, and financial losses.

At **Monethic**, we take a collaborative approach to security assessments, working closely with our clients to understand their specific needs and tailor our assessments accordingly. Our goal is to provide actionable recommendations and insights that help our clients make informed decisions about their security posture, while minimizing the risk of security incidents and financial losses.

The testers team consisted of:

- **Jakub Heba** - CEO and Co-Founder of Monethic - with over 8 years of experience in the cybersecurity industry. Holder of OSCP and OSCE certifications, have assigned multiple CVEs, among others:
    - **CVE-2019-10070** - Apache Atlas, Stored Cross Site Scripting
    - **CVE-2020-6856** - JOC Cockpit, Jobscheduler, XML External Entity
    - **CVE-2020-6854** - JOC Cockpit, Jobscheduler, Multiple Stored Cross Site Scripting
    - **CVE-2020-6855** - JOC Cockpit, Jobscheduler, Denial of Service
    - **CVE-2021-3584** - Foreman, Authenticated Remote Code Execution via Sendmail configuration

- **Łukasz Mikuła** - CTO and Co-Founder of Monethic - with over 8 years of experience in the cybersecurity industry. Holder of eWPT, eWPTX, OSCP and OSCE certifications, with assigned CVEs such as:
    - **CVE-2020-5907** – F5 TMOS Shell privilege escalation vulnerability,
    - **CVE-2018-6498, CVE-2018-6499, CVE-2018-6498, CVE-2018-6499** - Microfocus - AutoPass License Server Remote Code Execution,
    - **CVE-2017-10068, CVE-2018-2651, CVE-2018-2652, CVE-2018-2653, CVE-2018-2695** - BI Publisher, PeopleSoft Enterprise PeopleTools XSS, XXE, SSRF, XSLT execution,
    - **CVE-2017-1181, CVE-2017-1183, CVE-2017-1182** IBM TEP Server - SQL Injection, Authorization Bypass, OS Command Injection

Disclaimer

This report reflects a rigorous security assessment conducted on the specified product, utilizing industry-leading methodologies. While the service was carried out with the utmost care and proficiency, it is essential to recognize that no security verification can guarantee 100% immunity from vulnerabilities or risks.

Security is a dynamic and ever-evolving field. Even with substantial expertise, it is impossible to predict or uncover all future vulnerabilities. Regular and varied security assessments should be performed throughout the code development lifecycle, and engaging different auditors is advisable to obtain a more robust security posture.

This assessment is limited to the defined scope and does not encompass parts of the system or third-party components not explicitly included. It does not provide legal assurance of compliance with regulations or standards, and the client remains responsible for implementing recommendations and continuous security practices.

Scoping Details

The purpose of the assessment was to conduct a second Smart Contract Audit against BJustCoin Smart Contracts, shared with the Monethic through the GitHub platform and selected `9799bf8ff344230da93f4048c6ba3534d7bdcc12` commit hash.

Repository: https://github.com/BJustCoin/BJustCoin

The scope of the assessment includes the files listed below:

- `Bjustcoin.sol`
- `ICOManager.sol`
- `IVestingToken.sol`
- `Oracle.sol`
- `VestingManager.sol`
- `VestingToken.sol`

Timeframe

On December 4th 2024, an agreement was signed to conduct a penetration test of the Admin Dashboard web application, and BJustCoin Smart Contracts. The work began on December 5th.

On December 13th 2024, a report from the security assessment was shared with the Client.

On December 19th, the final report, including retest results was shared with the Client. The retested code was present in commit `69e1af07760d24ef7deb29ea1ff79b656f3acee6`.

VULNERABILITY CLASSIFICATION

All vulnerabilities described in the report were thoroughly classified in terms of the risk they generate in relation to the security of the contract implementation. Depending on where they occur, their rating can be estimated on the basis of different methodologies.

In most cases, the estimation is done by summarizing the impact of the vulnerability and its likelihood of occurrence. The table below presents a simplified risk determination model for individual calculations.

		Impact		
	Severity	**High**	**Medium**	**Low**
Likelihood	**High**	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Vulnerabilities that do not have a direct security impact, but may affect overall code quality, as well as open doors for other potential vulnerabilities, are classified as **INFORMATIONAL**.

Vulnerabilities summary

No.	Severity	Name	Status
1	Medium	Protocol may use Chainlink's stale price	**Resolved**
2	Medium	Not all Bjustcoin tokens can be burnt after ICO	**Resolved**
3	Low	Lack of two-step ownership transfer	**Resolved**
4	Low	Blacklisting does not take into account spender	**Resolved**
5	Low	The buyToken reverts for minimum sold amount	**Resolved**
6	Informative	Lack of event emission for key operations	**Resolved**
7	Informative	Private period sale behaves as public period	**Resolved**
8	Informative	Ownership renounce possible	**Resolved**

TECHNICAL SUMMARY

1. Protocol may use Chainlink's stale price

Severity

MEDIUM

Location

src/Oracle.sol

Description

The getLatestPrice returns the price feed's value without checking whether the price is stale. In the event when the stale price is returned the protocol may use underestimated or overestimated rate while selling tokens within the buyToken function. As a result an inaccurate number of tokens will be minted for vesting.

```
function getLatestPrice() public view returns (int256) {
    int256 price;
    (, price,,,) = priceFeed.latestRoundData();
    return price / 1e6;
}
```

Remediation

Whenever Chainlink's price feed is in use it is recommended to check its heartbeat and whether the returned price is not equal to 0. The heartbeat for the 0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419 feed is equal to 3600 seconds.

Any price value returned with the updatedAt not within the heartbeat should be considered stale. The protocol must individually decide what action should be taken in the event of the stale price occurrence.

Status: Resolved

2. Not all Bjustcoin tokens can be burnt after ICO

Severity

MEDIUM

Location

`src/Oracle.sol`

Description

Within the `nextICOStage` function, whenever the ICO period is ended all remaining, not sold `Bjustcoin` tokens are burnt, to decrease the total supply to the actual amount circulating. However, this function handles tokens only from `Strategic`, `Seed`, `PrivateSale`, `IDO`, and `PublicSale` sale periods.

Other tokens not sold during the `Advisors`, `Team`, `FutureTeam`, `Incentives`, `Liquidity`, `Ecosystem`, and `Loyalty` periods are not burnt after the sale period ends. Such residual tokens are locked within the `ICOManager` contract.

```
    function nextICOStage() external payable onlyOwner {
        //"ICO completed"
        if (icoStage == ICOStage.EndICO) {
            revert ICOCompleted();
        }
        ICOStage initialStage = icoStage;
        icoStage = ICOStage(uint256(icoStage) + 1);
        emit ICOStageChanged(msg.sender, initialStage, icoStage);
        if (icoStage != ICOStage.EndICO && initialStage != ICOStage.NoICO) {
            TokenomicType _initTokenomicType = getTokenomicType(initialStage);
            TokenomicType _tokenomicType = getTokenomicType(icoStage);

            tokenomicSettings[_tokenomicType].maxTokenCount += (
                tokenomicSettings[_initTokenomicType].maxTokenCount
                    - tokenomicSettings[_initTokenomicType].soldTokenCount
            );
            tokenomicSettings[_initTokenomicType].maxTokenCount =
 tokenomicSettings[_initTokenomicType].soldTokenCount;
            initVestingToken(tokenomicSettings[_tokenomicType]);
        } else if (icoStage == ICOStage.EndICO) {
            uint256 burnCount =
 tokenomicSettings[TokenomicType.PublicSale].maxTokenCount
                - tokenomicSettings[TokenomicType.PublicSale].soldTokenCount;
            tokenomicSettings[TokenomicType.PublicSale].maxTokenCount =
                tokenomicSettings[TokenomicType.PublicSale].soldTokenCount;
            _baseToken.burn(burnCount);
        } else {
            initVestingToken(tokenomicSettings[TokenomicType.Strategic]);
        }
    }
```

Remediation

It is recommended to review the defined business rules and protocol implementation and consider implementing additional functionality that burns not sold tokens from other sale periods.

Status: Resolved

---

3. Lack of two-step ownership transfer

Severity

LOW

Location

`src/ICOManager.sol`

Description

The protocol implements single-step ownership transfer since it is using OpenZeppelin's `Ownable` library. Thus, accidental transfer of ownership to unverified and incorrect address may result in loss of ownership.

In such a case, access to every function protected by the ownership check will be permanently lost.

```
contract ICOManager is Ownable {
```

Remediation

It is recommended to implement a two-step ownership transfer pattern within the solution, such as OpenZeppelin's `Ownable2Step` library instead of `Ownable` library.

Status: Resolved

---

4. Blacklisting does not take into account spender

Severity

LOW

Location

`src/Bjustcoin.sol`

Description

The Bjustcoin implements a blacklisting functionality that prevents token transfer `from` and `to` particular addresses. However, it does not address granting approvals and funds transfers done by spender.

Thus, an account with granted allowance, can still transfer funds between non-blacklisted addresses.

```
    function blacklist(address _address, bool _isBlacklisting) external onlyOwner {
        blacklists[_address] = _isBlacklisting;
    }
...
    function _update(address from, address to, uint256 value) internal virtual override {
        if (blacklists[to] || blacklists[from]) revert Blacklisted();
        super._update(from, to, value);
    }
```

Remediation

It is recommended to check whether the blacklisted spender attempts to call `transferFrom` function and revert it when identified.

Status: Resolved

5. The `buyToken` reverts for minimum sold amount

Severity

LOW

Location

`src/ICOManager.sol`

Description

The `buyToken` allows users to buy some vesting tokens in exchange for native tokens. It checks whether the minimum amount is provided. However, for the assertion checks it uses the <= comparison. Whenever the `msg.value` is equal to the minimum value, the transaction reverts, despite the fact the amount is actually sufficient.

```
function buyToken(TokenomicSetting storage settings) private {
    uint256 rate = getRate();
    if (msg.value <= MIN_SOLD_VOLUME * 1e18 / rate) {
        revert MinSoldError();
    }
```

Remediation

It is recommended to update the aforementioned assertion to use < instead of <=.

Status: Resolved

6. Lack of event emission for key operations

Severity

INFORMATIVE

Location

`src/Bjustcoin.sol`
`src/ICOManager.sol`

Description

The protocol implements blacklisting functionality, however, it does not emit any event upon blacklisting. Such omission may impact and hinder the off-chain processing and monitoring.

```
/**
  * @notice  Adding or removing an address to the blacklist
  * @dev     Adding or removing an address to the blacklist
  * @param   _address  Address additing or removing to the blacklist
  * @param   _isBlacklisting  true - add; false - remove;
  */
 function blacklist(address _address, bool _isBlacklisting) external onlyOwner {
     blacklists[_address] = _isBlacklisting;
 }

 /**
  * @notice  Adding or removing an address to the blacklist
  * @dev     Adding or removing an address to the blacklist
  * @param   _address  Address additing or removing to the blacklist
  * @param   _isBlacklisting  true - add; false - remov;
  */
 function blacklist(address _address, bool _isBlacklisting) external payable onlyOwner {
     if (blacklists[_address] != _isBlacklisting) {
         blacklists[_address] = _isBlacklisting;
         _baseToken.blacklist(_address, _isBlacklisting);
     }
 }
```

Remediation

It is recommended to implement event emission in the aforementioned functionality.

Status: Resolved

7. Private period sale behaves as public period

Severity

INFORMATIVE

Location

`src/ICOManager.sol`

Description

Apart from the Tokenomic Setting, there is no difference between private and public sale token periods. Usually, private token sale is protected by some sort of authorisation, such as whitelisting, Merkle Tree or signature-based authorisation.

However, in the case of a private period, whenever it is started anyone can buy tokens.

```
    function buyPrivateSaleToken() private {
        buyToken(tokenomicSettings[TokenomicType.PrivateSale]);
    }
...
    function buyPublicSaleToken() private {
        buyToken(tokenomicSettings[TokenomicType.PublicSale]);
    }
```

Remediation

It is recommended to review the defined business rules and protocol implementation and consider changing the approach taken for the private sale period.

Status: Resolved

8. Ownership renounce possible

Severity

INFORMATIVE

Location

`src/ICOManager.sol`

Description

The `ICOManager` contract implements the `Ownable` library. By default `Ownable` provides ownership renounce functionality. This function can be called accidentally and then, the owner address is set to 0 address. As a result, all functionalities protected by the `onlyOwner` modifier, such as `withdraw`, are permanently locked.

Remediation

It is recommended to override and disallow the ownership renounce functionality to be successfully called.

Status: Resolved

END OF THE REPORT