SALUS SECURITY

MAY 2023

CODE SECURITY ASSESSMENT

MULTIBIT

Overview

Project Summary

- Name: MultiBit
- Version: commit 221e701
- Platform: EVM-compatible chains
- Language: Solidity
- Repository: https://github.com/multibit-repo/multibit
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	MultiBit
Version	v2
Type	Solidity
Date	May 31 2023
Logs	May 26 2023; May 31 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	4
Total informational issues	7
Total	13

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Circulating supply of MultiBit token will be different than totalSupply	Medium	Business logic	Resolved
2	Missing duplicate checks for signers in the constructor	Medium	Business logic	Acknowledged
3	Missing zero-length check for array parameters in mint()	Low	Data Validation	Acknowledged
4	Lack of expiration time for signatures in mint()	Low	Data Validation	Acknowledged
5	Surplus of msg.value is not returned if sent more	Low	Business logic	Acknowledged
6	Validate array length matching before execution to avoid reverts	Low	Data Validation	Acknowledged
7	Use of payable.transfer() might render ETH impossible to withdraw	Informational	Business logic	Acknowledged
8	Use of floating pragma	Informational	Configuration	Acknowledged
9	CREATE2 implementation does not allow deploying different versions of BRC20 contract	Informational	Business logic	Acknowledged
10	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged
11	Race condition for approve()	Informational	Front-running	Acknowledged
12	Redundant code	Informational	Redundancy	Acknowledged
13	Gas optimization suggestions	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Circulating supply of MultiBit token will be different than totalSupply	
Severity: Medium	Category: Business logic
Target: - src/MultiBitToken.sol	

Description

The MultiBitToken contract declares a totalSupply variable to track the totalSupply of the MultiBit token. This totalSupply variable is meant to be set to the initial supply of tokens minted to the deployer when the contract is deployed.

src/MultiBitToken.sol:L10

```
uint256 public totalSupply = 100000000 * 10**18;
```

src/MultiBitToken.sol L22-L30

```
constructor() {
       _mint(msg.sender, totalSupply);
       uint256 chainId;
       assembly {
           chainId := chainid()
       }
       DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)),
keccak256(bytes('1')), chainId, address(this)));
}
```

However, since the _mint() function increases the totalSupply state variable, the value returned from totalSupply() will be twice the actual amount supplied when the contract is deployed.

src/MultiBitToken.sol:L32-L41

```
function _mint(address to, uint256 amount) internal {

       totalSupply += amount;

       unchecked {
           balanceOf[to] += amount;
       }

       emit Transfer(address(0), to, amount);
}
```

Recommendation

Design a separate variable (e.g. initialSupply) to represent the initial supply of tokens minted to the deployer and use it in the constructor's _mint() function instead.

Status

This issue has been resolved by the team in commit 8e622f1. The team has removed the assignment at line 10, and a hardcoded amount of tokens are minted in the constructor.

2. Missing duplicate checks for signers in the constructor

Severity: Medium	Category: Business Logic
Target: - src/BRC20Factory.sol	

Description

There is no check for ensuring that there are no duplicates in _signers array passed in the constructor. So, if there are duplicates in the signers array, mint functionality won't work because it checks if the signature is provided by all the signers and there are no duplicate signers.

src/BRC20Factory.sol:L56-L74

```
constructor(address[] memory _signers) {
       for (uint256 i = 0; i < _signers.length; i++) {
           address _addr = _signers[i];
           signers.push(_addr);
           authorized[_addr] = true;
           indexes[_addr] = i;
       }

       owner = msg.sender;
       emit OwnerChanged(address(0), msg.sender);

       fee = 0.01 ether;

       uint256 chainId;
       assembly {
           chainId := chainid()
       }
       DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, DOMAIN_NAME,
keccak256(bytes('1')), chainId, address(this)));
   }
```

src/BRC20Factory.sol:L91-98

```
for (uint256 i = 0; i < v.length; i++) {
       address signer = ecrecover(digest, v[i], r[i], s[i]);
       require(authorized[signer], "invalid signer");
       for (uint256 j = 0; j < i; j++) {
           require(signatures[j] != signer, "duplicated");
       }
       signatures[i] = signer;
}
```

Recommendation

Consider adding a check to ensure that there are no duplicates in the _signers array passed in the constructor.

Status

This issue has been acknowledged by the team.

3. Missing zero-length check for array parameters in mint()	
Severity: Low	Category: Data Validation
Target: - src/BRC20Factory.sol	

Description

The mint() function in the BRC20Factory contract is not checking whether the passed-in v argument is an empty array. If the owner mistakenly removes all the signers, users can pass an empty v array and bypass the following check, allowing them to free mint tokens.

src/BRC20Factory.sol:L91-L98

```
for (uint256 i = 0; i < v.length; i++) {
        address signer = ecrecover(digest, v[i], r[i], s[i]);
        require(authorized[signer], "invalid signer");
        for (uint256 j = 0; j < i; j++) {
                require(signatures[j] != signer, "duplicated");
        }
        signatures[i] = signer;
}
```

Recommendation

Consider adding a zero-length check for v.length in mint() function.

Status

This issue has been acknowledged by the team.

4. Lack of expiration time for signatures in mint()	
Severity: Low	Category: Data Validation
Target: - src/BRC20Factory.sol	

Description

The mint() function does not set an expiration time for digests and signatures. If a transaction remains in the mempool for a long time, it could be executed much later than the signer anticipated.

Recommendation

Consider including an expiration timestamp in the signed digest and checking whether the mint() transaction has expired.

Status

This issue has been acknowledged by the team.

5. Surplus of msg.value is not returned if sent more	
Severity: Low	Category: Business logic
Target: - src/BRC20Factory.sol	

Description

There is only a check that msg.value should be greater than or equal to the fee value. If a user mistakenly sent more msg.value than fee there is no surplus returned to the user.

src/BRC20Factory.sol:L105-L112

```
function burn(address token, uint256 amount, string memory  receiver) external payable
nonReentrant {
        require(msg.value >= fee, "invalid ether");

        BRC20(token).transferFrom(msg.sender, address(this), amount);
        BRC20(token).burn(amount);

        emit Burned(token, msg.sender, amount, fee, receiver);
}
```

Recommendation

Consider changing `msg.value >= fee` to `msg.value == fee`.

Status

This issue has been acknowledged by the team.

6. Validate array length matching before execution to avoid reverts	
Severity: Low	Category: Data Validation
Target: - src/BRC20Factory.sol	

Description

BRC20Factory contract has a mint() function that accepts multiple arrays that contain the necessary data for execution. Currently, it only checks if v.length equals signers.length. However, other arrays need to be of the same length because individual elements in the arrays are intended to be matched at the same indices.

Recommendation

Implement a check on the array lengths so they match.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

7. Use of payable.transfer() might render ETH impossible to withdraw	
Severity: Informational	Category: Business logic
Target: - src/BRC20Factory.sol	

Description

The protocol uses Solidity's transfer() in withdraw() function to transfer ETH. transfer() forward exactly 2300 gas to the recipient. The goal of this hardcoded gas stipend was to prevent reentrancy vulnerabilities, but this only makes sense under the assumption that gas costs are constant. Recently EIP 1884 was included in the Istanbul hard fork. One of the changes included in EIP 1884 is an increase to the gas cost of the SLOAD operation, causing a contract's fallback function to cost more than 2300 gas.

src/BRC20Factory.sol:L114-L117

```
function withdraw(address to) external {
        require(msg.sender == owner, "unauthorized");
        uint256 balance = address(this).balance;
        payable(to).transfer(balance);
}
```

Recommendation

Consider using address.call{value: balance}("") with its returned boolean checked in combination with re-entrancy guard is highly recommended.

Status

This issue has been acknowledged by the team.

8. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - src/BRC20.sol - src/MultiBitToken.sol - src/BRC20Factory.sol	

Description

```
pragma solidity ^0.8.0;
```

The MultiBit contracts use a floating compiler version ^0.8.0.

Using a floating pragma ^0.8.0 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

9. CREATE2 implementation does not allow deploying different versions of BRC20 contract

Severity: Informational	Category: Business logic
Target: - src/BRC20Factory.sol	

Description

The createBRC20() function does not have a parameter for making salt pseudorandomness. This means if the owner is going to deploy a new version of BRC20 with the same name, symbol and decimals, the transaction will fail.

src/BRC20Factory.sol:L70-L76

```
function createBRC20(string memory name, string memory symbol, uint8 decimals) external
returns (address brc20) {
        require(msg.sender == owner, "unauthorized");
        parameters = Parameters({name: name, symbol: symbol, decimals: decimals});
        brc20 = address(new BRC20{salt: keccak256(abi.encode(name, symbol,
decimals))}());
        delete parameters;
        emit BRC20Created(msg.sender, brc20);
}
```

Recommendation

Consider adding an additional parameter during deployment over CREATE2.

Status

This issue has been acknowledged by the team.

10. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - src/BRC20Factory.sol	

Description

The BRC20Factory contract uses a custom function setOwner() which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

src/BRC20Factory.sol:L119-L122

```
function setOwner(address _owner) external onlyOwner {
        emit OwnerChanged(owner, _owner);
        owner = _owner;
}
```

Recommendation

It is recommended to implement a two-step transfer of ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout.

Status

This issue has been acknowledged by the team.

11. Race condition for approve()	
Severity: Informational	Category: Front-running
Target: - src/BRC20.sol - src/MultiBitToken.sol	

Description

Since there is no direct way to increase and decrease allowance relative to its current value, the function approve(address,uint256) has a race condition similar to one of ERC-20 approvals. Further details regarding the race condition can be found here.

Simply put, the approve() function creates the potential for an approved spender to spend more than the intended amount. A front running attack can be used to enable an approved spender to call transferFrom() both before and after the call to approve() is processed.

Recommendation

Consider adding increaseAllowance() and decreaseAllowance() functions for BRC20 and MultiBitToken tokens, similar to what OpenZeppelin did with its ERC-20 implementation.

Status

This issue has been acknowledged by the team.

12. Redundant code	
Severity: Informational	Category: Redundancy
Target: - src/BRC20Factory.sol	

Description

The variable receiver is a redundant input variable in the burn() function. It is only used for emitting events. The caller can pass any address and consequently emit event with the passed receiver address.

src/BRC20Factory.sol:L100-L107

```
function burn(address token, uint256 amount, string memory receiver) external payable
nonReentrant {
       require(msg.value >= fee, "invalid ether");

       BRC20(token).transferFrom(msg.sender, address(this), amount);
       BRC20(token).burn(amount);

       emit Burned(token, msg.sender, amount, fee, receiver);
}
```

In the removeSigner() function. If an account is a signer, its index is less than signers.length. If there is no signer, the check at line 140 will fail. Thus, line 141 is redundant.

src/BRC20Factory.sol:L140-L141

```
require(authorized[account], "non-existent");
require(indexes[account] < signers.length, "index out of range");
```

Recommendation

Consider deleting receiver as input parameter and from emitting event, and removing the redundant code.

Status

This issue has been acknowledged by the team.

13. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - src/BRC20.sol - src/MultiBitToken.sol - src/BRC20Factory.sol	

Description

src/BRC20Factory.sol:L13

```
bytes32 public DOMAIN_SEPARATOR;
```

The state variable DOMAIN_SEPARATOR could be declared as immutable since its value is fixed after the contract has been deployed.

src/MultiBitToken.sol:L7-L8

```
string public name = "MultiBit Token";
string public symbol = "MUBI";
```

The state variable name and symbol could be declared as constant since its value is fixed before the contract compilation.

src/BRC20Factory.sol:L76

```
function createBRC20(string memory name, string memory symbol, uint8 decimals) external
onlyOwner returns (address brc20)
```

src/BRC20Factory.sol:L83

```
function mint(address token, address to, uint256 amount, string memory txid, uint8[]
memory v, bytes32[] memory r, bytes32[] memory s) external nonReentrant
```

src/BRC20Factory.sol:L159

```
function buildMintSeparator(address token, address to, uint256 amount, string memory
txid) view public returns (bytes32)
```

Mark data types as calldata instead of memory if the data passed into the function does not need to be changed.

Recommendation

Consider making changes based on the above suggestions.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 221e701:

File	SHA-1 hash
src/BRC20.sol	44abcfa25624a58e290f7cc3f5fe3a02fd5eb74f
src/interfaces/IBRC20.sol	4e0b5b09512bcf913e48aca6c07a30f072c31136
src/interfaces/IBRC20Factory.sol	152ce2dcfb1c1fbebbb966d385bc80dfca80a23b
src/MultiBitToken.sol	46defbf9ae24896d4289d2f1cdea83f1be2405c8
src/BRC20Factory.sol	dcee459be216eb6ba4a45e9645d373ae2d8d7c9c