SALUS SECURITY

MAY 2023

CODE SECURITY ASSESSMENT

BINARYX

Overview

Project Summary

- Name: BinaryX - Matthew
- Platform: BNB Smart Chain
- Language: Solidity
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	BinaryX - Matthew
Version	v2
Type	Solidity
Date	May 26 2023
Logs	May 16 2023; May 26 2023

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	3
Total	7

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	_batchIds not updated in the addBatch() function	High	Business logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Hardcoded addresses	Low	Configuration	Acknowledged
4	Return value of ERC20's transfer()/transferFrom() not checked	Low	Validation	Resolved
5	Missing validations	Informational	Validation	Acknowledged
6	Gas optimization recommendations	Informational	Gas inefficiencies	Acknowledged
7	Floating compiler version	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. _batchIds not updated in the addBatch() function

Severity: High	Category: Business logic
Target: - Land.sol	

Description

Land.sol:L34-L41

```
function tokenURI(uint256 tokenId) public view override returns (string memory) {
    require(_exists(tokenId), "Land: URI query for nonexistent token");

    string memory baseURI = _baseURI();
    return bytes(baseURI).length > 0 ?
    string(abi.encodePacked(baseURI, _batchIds[tokenId / BATCH_SIZE], "/",
tokenId.toString())) :
    "";
}
```

The tokenURI() function of the Land contract relies on the _batchIds state variable to determine the token URI for a given tokenId when the baseURI is set. However, the addBatch() function does not update _batchIds, leaving it an empty array. Thus, _batchIds[tokenId / BATCH_SIZE] throws an index-out-of-bounds error, rendering the tokenURI() function unusable.

Recommendation

We recommend updating the _batchIds state variable in the addBatch() function accordingly.

Status

The team has resolved this issue by adding the following line in addBatch():

```
_batchIds.push(batchId);
```

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - all	

Description

The Matthew project has a privileged OPERATOR account. This account:

- is the initial signer of the PaymentManager, WalletManagerV2, LandUpgrader, LandVoucherMinter, and RebateKeeper contracts
- is the initial signer of the ETH, BNX, MATTHEW_COIN and LAND wallets
- has the ROLE_OPERATOR role in the PaymentManager, WalletManagerV2, MatthewCoin, LandVoucher, SoldierVoucher, Land, LandUpgrader, LandVoucherMinter and RebateKeeper contracts

The deployer account of the Deployer contract is another privileged account (let's call it OWNER). The OWNER account:

- is the owner of the PaymentManager, WalletManagerV2, MatthewCoin, LandVoucher, SoldierVoucher, Land, LandUpgrader, LandVoucherMinter, and RebateKeeper contracts.
- can act as the ROLE_DEPLOYER and ROLE_OPERATOR roles in those contracts

If either the OPERATOR or OWNER private key is compromised, an attacker can:

- drain all the funds from the wallet contracts
- mint Land, LandVoucher, MatthewCoin, SoldierVoucher tokens
- burn LandVoucher, MatthewCoin, SoldierVoucher tokens

If these privileged accounts are plain EOA accounts, this poses a risk to the users. A compromised private key could be used to exploit the privileged operations and attack the project.

Recommendation

We recommend transferring privileged accounts to multi-signature accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

3. Hardcoded addresses

Severity: Low	Category: Configuration
Target: - Deployer.sol	

Description

Deployer.sol:L33-L45

```
constructor() {
    uint256 chainId;
    assembly { chainId := chainid() }

    if (chainId == 97) {
        RECIPIENT = 0xc9fc6362e6627bC981368C5284fe71de814408BB;
        OPERATOR = 0x1d642bA5Bd1c4b6d669310E9ECf874C429f11960;
        BNX = 0x466d8464536d36ADA0334FE54E3De776FB7181eA;
    } else {
        RECIPIENT = address(0);
        OPERATOR = address(0);
        BNX = address(0);
    }
    ...
}
```

The RECIPIENT, OPERATOR, and BNX addresses are hardcoded in the constructor() function of the Deployer contract. Moreover, they are only set when the contract is deployed to the BSC testnet (chainId == 97).

If the developer forgets to change the hardcoded values before deploying to the BSC mainnet, the RECIPIENT, OPERATOR, and BNX will be set to the zero-address, which will cause the project to malfunction.

Recommendation

We recommend that instead of using hardcoded addresses, the developer could input the addresses as parameters for the constructor() function. Moreover, the developer should ensure that the addresses are configured correctly before deploying the contracts to the mainnet.

Status

This issue has been acknowledged by the team.

4. Return value of ERC20's transfer()/transferFrom() not checked

Severity: Low	Category: Validation

Target:
- ERC20Keeper
- PaymentManager.sol
- WalletManagerV2.sol

Description

Not all IERC20 implementations revert when there's a failure in transfer()/transferFrom(). The function signature has a boolean return value and they indicate errors that way instead. By not checking the return value, operations that should have been marked as failed, may potentially go through without actually making the transfer.

We have found 5 instances of this issue:
ERC20Keeper.sol:L9

```
IERC20(token).transfer(to, amount);
```

PaymentManager.sol:L57

```
IERC20(tokenAddr).transferFrom(account, recipient, amount);
```

WalletManagerV2.sol:L28

```
IERC20(tokenAddr).transfer(to, amount);
```

WalletManagerV2.sol:L75

```
IERC20(_tokenAddr).transferFrom(account, address(this), amount);
```

WalletManagerV2.sol:L87

```
IERC20(_tokenAddr).transfer(account, amount);
```

Recommendation

We recommend using OpenZeppelin's SafeERC20 with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

Status

This issue has been resolved by the team.

2.3 Informational Findings

5. Missing validations

Severity: Informational	Category: Validations

Target:
- BasicAccessControl.sol
- Deployer.sol
- UseSigner.sol
- PaymentManager.sol
- WalletManagerV2.sol
- LandUpgrader.sol
- LandVoucherMinter.sol
- RebateKeeper.sol

Description

There are some places in the code base that might benefit from some sanity checks on the input provided:

- The grantDeployer(), revokeDeployer(), grantOperator(), and revokeOperator() functions of the BasicAccessControl contract are not checking the account address to be non-zero.
- The addDeployer() function of the Deployer contract is not checking the deployer address to be non-zero.
- The setSigner() function of the UseSigner contract is not checking the newSigner address to be non-zero.
- The setRecipients() function of the PaymentManager is not checking the addr address to be non-zero.
- The constructor() functions of PaymentManager, WalletManagerV2, LandUpgrader, LandVoucherMinter, and RebateKeeper are missing the same check over the address parameter.

Moreover:

- The burnForVoucher() function of the LandVoucherMinter contract is not checking the length of the tokenIds input array to be non-zero.

Recommendation

To reduce possible errors and make the code more robust, consider adding sanity checks where needed.

Status

This issue has been acknowledged by the team.

6. Gas optimization recommendations

Severity: Informational	Category: Gas inefficiencies
Target: - all	

Description

Some changes can be made to improve gas consumption:

1. For functions that are only called from outside the smart contract, using function visibility **public** can make the gas costs greater than using the visibility **external**. This is because with public functions, the EVM copies inputs (especially dynamic-sized arrays) into memory, while it reads from calldata if the function is external, which is cheaper. Consider changing the visibility of public functions to **external** if they are only accessed externally. For example, the addDeployer() function of the Deployer contract.
2. Variables only set in the constructor and never edited afterwards should be marked as **immutable**, as it would avoid the expensive storage-writing operation in the constructor and replace the expensive storage-reading operations to a less expensive value reading. Consider defining the state variables as immutable if they won't be changed after deployment. For example, the BNX variable of the Deployer contract.

Recommendation

Consider applying the gas optimizations where needed.

Status

This issue has been acknowledged by the team.

7. Floating compiler version

Severity: Informational	Category: Configuration
Target: - all	

Description

```
pragma solidity ^0.8.0;
```

The Matthew code base uses a floating Solidity compiler version, ^0.8.0.

It’s best to deploy contracts with the same compiler version and flags that they have been thoroughly tested with. Locking the compiler version helps to prevent contracts from being accidentally deployed using an outdated compiler version, which could introduce bugs that have a negative impact on the system.

Recommendation

It is recommended to use a locked Solidity compiler version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in the matthew.zip file provided by the client:

File	SHA-1 hash
BasicAccessControl.sol	7b8c2141116c1cb0e497f40be3568548be2e4555
Bytes.sol	e0ce69ae6ba607af858aac9b99bf9ad0bd494de2
Deployer.sol	2a14e3f1f4a0657bfabe9eb4ab2c0fd151d3be92
ERC20Keeper.sol	35be8cd74007bd69b2c6cc05f07325542e9d988b
EtherKeeper.sol	fe7b6b6dde7991bb95959e62b72679171b1a7b09
EtherReceiver.sol	dd1661a9d04f9bf9d23e4cc2ba9bce2c5ecc7be5
IMatthewV1.sol	7bf12d9281b58d2cd0025656037f4309eedc3507
Land.sol	72d1ff090642180fe18c15b7e6ffe11ca111d893
LandUpgrader.sol	6b8ab327ca376d198cf01b44eb95961e17d3ce3e
LandVoucher.sol	eb9962140c736605b59689293f8d040b1b52989b
LandVoucherMinter.sol	8264352ebadc718375451c000ada68ee393c20cc
MatthewCoin.sol	6c228596c520f653b0230dbf38858a7cc4330d09
PaymentManager.sol	078e598343498195d09ca2c1e769681b331948d1
RebateKeeper.sol	7a02c2e971ce22c9e15be87fc3d524d1b61b7cd9
RequestTracing.sol	95d22a67a0adf474a7fb69eb0a1eb2b8862d403d
SignatureVerifier.sol	4d138b50095a45947a43b4b5fa3e6fedfe23a8f6
SoldierVoucher.sol	829f18c4fdbbe511b28e3f9e034b860b16b59faa
UseSigner.sol	4030bd47ff993c1c3ed13bebf127780b88bb2adf
WalletManagerV2.sol	3a911e4e901be61ee43186f5482d74f00d8976e8