SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

BIGA

Overview

Project Summary

- Name: BIGA - Arcade Incremental Audit
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/bigaarcade/arcade-sc
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	BIGA - Arcade Incremental Audit
Version	v2
Type	Solidity
Dates	Oct 14 2024
Logs	Sep 24 2024, Oct 14 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	1
Total	6

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	Missing withdrawnTotal reset	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Hard fork can potentially lead to the reply attacks	Medium	Business Logic	Resolved
4	Use call instead of transfer for native tokens transfer	Low	Business Logic	Resolved
5	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
6	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

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

1. Missing withdrawnTotal reset	
Severity: Medium	Category: Business Logic
Target: - contracts/BIGA.sol	

Description

The contract imposes a limit on withdrawals during each window, ensuring that the total amount withdrawn does not surpass the `withdrawalLimitAmount`. This limit is calculated based on the contract's token balance and the amount already withdrawn within the current withdrawal period.

However, when the withdrawal window resets, the `withdrawnTotal` from the previous window remains intact. As a result, the `withdrawalLimitAmount` for the first transaction in the new window may exceed expectations.

contracts/BIGA.sol:L187-L194

```
function withdraw(
    address _tokenIn,
    address _tokenOut,
    uint256 _amountOut,
    uint256 _nonce,
    bytes calldata _signature
) external nonReentrant {
    ...
    if (checkWindow()) {
        require(withdrawnTotal[_tokenOut] + _amountOut <=
_withdrawalLimitAmount(_tokenOut), "GB08");
        withdrawnTotal[_tokenOut] += _amountOut;
    } else {
        require(_amountOut <= _withdrawalLimitAmount(_tokenOut), "GB08");
        withdrawnTotal[_tokenOut] = _amountOut;
        windowStartTime = block.timestamp;
    }
    ...
}
```

Recommendation

We recommend resets the `withdrawTotal` during the window restart.

Status

This issue has been resolved by the team with commit 2710122.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - contracts/BIGA.sol	

Description

In the `BIGA` contract, there exists a privileged role called `owner`. The `owner` has the authority to modify critical parameters in the contract, such as `validator`, `withdrawalLimit`, `windowDuration`, and the contract's `tokenWhitelist`.

If the `owner`'s private key is compromised, an attacker could modify these parameters to steal tokens from the contract.

Recommendation

We recommend transferring privileged accounts to multi-sig 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. The team states that once the contract is deployed, the privileged role will be assigned to a multi-sig account.

3. Hard fork can potentially lead to replay attacks	
Severity: Medium	Category: Configuration
Target: - contracts/BIGA.sol	

Description

The contract stores the `chainId` used for signature verification in a global variable, which can only be initialized once. When a hard fork occurs, the `chainId` stored in the contract will no longer match the current `chainId`.This introduces the risk of replay attacks.

contracts/BIGA.sol:L226-L241

```
function _validateWithdrawData(
    bytes calldata _signature,
    address _user,
    address _tokenIn,
    address _tokenOut,
    uint256 _amountOut,
    uint256 _nonce
) private {
    bytes memory data = abi.encodePacked(chainId, _user, _tokenIn, _tokenOut,
_amountOut, _nonce);
    data.verifySignature(_signature, validator, "GB04");

    bytes32 dataHash = keccak256(data);
    require(!hashUsed[dataHash], "GB05");

    hashUsed[dataHash] = true;
}
```

Recommendation

We recommend not storing the `chainId` and instead retrieving it in real-time during calculations.

Status

This issue has been resolved by the team with commit 2710122.

4. Use call instead of transfer for native tokens transfer	
Severity: Informational	Category: Business logic
Target: - contracts/BIGA.sol	

Description

The transfer function is not recommended for sending native tokens due to its 2300 gas unit limit which may not work with smart contract wallets or multi-sig. Instead, call can be used to circumvent the gas limit.

contracts/BIGA.sol:L197-L203

```
function withdraw(
    address _tokenIn,
    address _tokenOut,
    uint256 _amountOut,
    uint256 _nonce,
    bytes calldata _signature
) external nonReentrant {
    ...
    // Withdraw token to user
    if (_tokenOut == address(0)) {
        payable(user).transfer(_amountOut);
    } else {
        _tokenOut.checkERC20("GB02");
        IERC20(_tokenOut).safeTransfer(user, _amountOut);
    }

    emit Withdrawn(user, _tokenIn, _tokenOut, _amountOut, _nonce, chainId);
}
```

Recommendation

Consider using call instead of transfer for sending native token.

Status

This issue has been resolved by the team with commit 2710122.

5. Implementation contract could be initialized by everyone	
Severity: Low	Category: Business Logic
Target: - contracts/BIGA.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the initialize function in BIGA's implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the BIGA contract to automatically lock it when it is deployed.

Status

This issue has been resolved by the team with commit 2710122.

2.3 Informational Findings

6. Redundant Code	
Severity: Informational	Category: Redundancy
Target: - contracts/BIGA.sol	

Description

There are unnecessary checks in the `removeFromWhitelist` function. Since the token address is already checked as ERC20 during addition, and if a token in the whitelist is later updated in a way that causes `checkERC20` to revert, it would prevent the non-compliant token from being removed from the whitelist.

contracts/BIGA.sol:L119-L128

```
function removeFromWhitelist(address[] calldata _tokens) external onlyOwner {
    for (uint i = 0; i < _tokens.length; i++) {
        if (_tokens[i] != address(0)) {
            _tokens[i].checkERC20("GB02");
        }
        tokenWhitelist[_tokens[i]] = false;
    }

    emit TokenRemovedFromWhitelist(_tokens);
}
```

Recommendation

We recommend removing the `checkERC20` check from the `removeFromWhitelist` function.

Status

This issue has been resolved by the team with commit 2710122.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit b005181:

File	SHA-1 hash
BIGA.sol	f626000ced4c9a3932f233efdade6ffbe3f64537
SafeCheck.sol	d29f13a822681f26632d6b9cb5ab0f320bca5404
VerifySignature.sol	14c3471d18c40236b18b273c70439bed7ab8550c