SALUS SECURITY

JAN 2025

CODE SECURITY ASSESSMENT

BID

Overview

Project Summary

- Name: Bid
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Bid
Version	v2
Type	Solidity
Dates	Jan 06 2025
Logs	Jan 03 2025; Jan 06 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	2
Total	4

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	Checking errors in target versus actual quantities	High	Business Logic	Resolved
2	Centralization Risk	Low	Centralization	Acknowledged
3	Redundant code	Informational	Redundancy	Resolved
4	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged

2.2 Notable Findings

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

1. Checking errors in target versus actual quantities

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

Description

The `purchase()` function is used for buying and selling between `usdc` and `bid` tokens, where there exists a target value of `usdc`, `_usdcToCollect`, which stops the function from running when the `usdc` in the contract reaches the target value. However, there is a serious problem with the logic of checking whether the target value has been reached.

If the difference between the current target value and the actual value is only 1 ether, but the user passes in a value of 2 ether for this purchase, it will cause the actual value to exceed the target value, but it will not terminate the function, it will continue to run normally. This is because the function uses `==` as a judgement condition.

BIDPublicSale.sol:L112 - L135

```
function purchase(
    bytes calldata _signature,
    uint256 _signatureExpiresAt,
    uint256 _amount
) external nonReentrant {
    if (!hasStarted || usdcCollected == _usdcToCollect) revert Errors.Unavailable();
    if (_amount == 0) revert Errors.InvalidAmount();
    _verifySignature(_signature, _signatureExpiresAt);
    if (
        usdc.balanceOf(msg.sender) < _amount ||
        usdc.allowance(msg.sender, address(this)) < _amount
    ) revert Errors.InsufficientFunds();
    if (_maxWallet < sale[msg.sender].amount + _amount) revert Errors.InvalidAmount();
    sale[msg.sender].amount += _amount;
    uint256 bidAmount = (_bidToSell * _amount) / _usdcToCollect;
    usdcCollected += _amount;
    if (usdcCollected == _usdcToCollect) {
        bidAmount = _bidToSell - bidSold;
    }
    bidSold += bidAmount;
    usdc.safeTransferFrom(msg.sender, address(this), _amount);
    bid.safeTransfer(msg.sender, bidAmount);
    emit Purchased(msg.sender, _amount, bidAmount);
}
```

Recommendation

Consider changing `==` to `>=` and adding `_amount` into the conditional logic.

Status

The team has resolved this issue.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - CreatorBid.sol	

Description

The `CreatorBid` contract contains `miner` and `owner` privileged addresses. The `minter` has the right to mint a number of tokens up to the maximum limit for any address, and the `owner` has the ability to add and remove any address from the whitelist. In addition, the `owner` has the right to modify the minter.

If the private key of the `minter` or `owner` is compromised, an attacker could mint a large number of tokens for himself or arbitrarily remove the address whitelist to disrupt the normal operation of the protocol.

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.

2.3 Informational Findings

3. Redundant code

Severity: Informational	Category: Redundancy
Target: - BIDPublicSale.sol	

Description

The `Sale` struct in the `BIDPublicSale` contract is used to store relevant information about a sale. However, the `hasReceivedBID` field is not yet utilized, resulting in data redundancy.

BIDPublicSale.sol:L23 - 27

```
struct Sale {
    /// @dev usdc amount
    uint256 amount;
    bool hasReceivedBID;
}
```

Recommendation

Consider removing the `hasReceivedBID` field.

Status

The team has resolved this issue.

4. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - BIDPublicSale.sol - CreatorBid.sol	

Description

The `BIDPublicSale` and `CreatorBid` contracts inherit from the `Ownable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
BIDPublicSale.sol	36ee23e4dfaae928bfd1a40850a2929dfaf9c98b
CreatorBid.sol	f2dbf08b9c9f63a1b16a20f32fe033a84b2a3d0f

And we audited the fixed version of the project:

File	SHA-1 hash
BIDPublicSale.sol	f52e8577ec7f293b562b2295ba9ad8461fc225a4