Overview

Project Summary

● Name: BOUNCE
● Version: v3
● Platform: EVM-compatible chains
● Language: Solidity
● Audit Range: See Appendix - 1

Project Dashboard

Application Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>BOUNCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>v2</td>
</tr>
<tr>
<td>Type</td>
<td>Solidity</td>
</tr>
<tr>
<td>Dates</td>
<td>Feb 01 2023</td>
</tr>
<tr>
<td>Logs</td>
<td>Jan 29 2023; Feb 01 2023</td>
</tr>
</tbody>
</table>

Vulnerability Summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total High-Severity issues</td>
<td>0</td>
</tr>
<tr>
<td>Total Medium-Severity issues</td>
<td>1</td>
</tr>
<tr>
<td>Total Low-Severity issues</td>
<td>5</td>
</tr>
<tr>
<td>Total informational issues</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>11</td>
</tr>
</tbody>
</table>

Contact

E-mail: support@salusec.io
## Risk Level Description

<table>
<thead>
<tr>
<th>Risk Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High Risk</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Medium Risk</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Low Risk</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Informational</strong></td>
<td>The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.</td>
</tr>
</tbody>
</table>
Content

Introduction

1.1 About SALUS 4
1.2 Audit Breakdown 4
1.3 Disclaimer 4

Findings

2.1 Summary of Findings 5
2.2 Notable Findings 6
   1. Missing storage gap for upgradeable contracts 6
   2. Message call with hardcoded gas amount 7
   3. Lack of protection against signature replay attacks 9
   4. Unnecessary comment of the priceHash implementation 10
   5. Single-step ownership transfer pattern risk 12
   6. Missing check in initialization function 13
2.3 Informational Findings 14
   7. Floating compiler version 14
   8. Redundant code 15
   9. SafeMath library not needed since Solidity 0.8.0 16
10. Lack of NatSpec documentation 17
11. External call to an out of scope address 18

Appendix

Appendix 1 - Files in Scope 19
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

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Severity</th>
<th>Category</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Missing storage gap for upgradeable contracts</td>
<td>Medium</td>
<td>Business Logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>2</td>
<td>Message call with hardcoded gas amount</td>
<td>Low</td>
<td>Business Logic</td>
<td>Resolved</td>
</tr>
<tr>
<td>3</td>
<td>Lack of protection against signature replay attacks</td>
<td>Low</td>
<td>Cryptography</td>
<td>Resolved</td>
</tr>
<tr>
<td>4</td>
<td>Unnecessary comment of the priceHash implementation</td>
<td>Low</td>
<td>Redundancy</td>
<td>Resolved</td>
</tr>
<tr>
<td>5</td>
<td>Single-step ownership transfer pattern risk</td>
<td>Low</td>
<td>Authentication</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>6</td>
<td>Missing check in initialization function</td>
<td>Low</td>
<td>Data Validation</td>
<td>Resolved</td>
</tr>
<tr>
<td>7</td>
<td>Floating compiler version</td>
<td>Informational</td>
<td>Configuration</td>
<td>Resolved</td>
</tr>
<tr>
<td>8</td>
<td>Redundant code</td>
<td>Informational</td>
<td>Redundancy</td>
<td>Unresolved</td>
</tr>
<tr>
<td>9</td>
<td>SafeMath library not needed since Solidity 0.8.0</td>
<td>Informational</td>
<td>Redundancy</td>
<td>Resolved</td>
</tr>
<tr>
<td>10</td>
<td>Lack of NatSpec documentation</td>
<td>Informational</td>
<td>Code Quality</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>11</td>
<td>External call to an out of scope address</td>
<td>Informational</td>
<td>Undefined</td>
<td>Acknowledged</td>
</tr>
</tbody>
</table>
2.2 Notable Findings

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

### 1. Missing storage gap for upgradeable contracts

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Category: Business Logic</th>
</tr>
</thead>
</table>

**Target:**
- contracts/BounceBase.sol
- contracts/BounceDutchAuction.sol
- contracts/BounceFixedSwap.sol
- contracts/BounceSealedBid.sol

**Description**

`contracts/BounceDutchAuction.sol:7`

```solidity
contract BounceDutchAuction is BounceBase
```

`contracts/BounceFixedSwap.sol:L7`

```solidity
contract BounceFixedSwap is BounceBase
```

`contracts/BounceSealedBid.sol:L7`

```solidity
contract BounceSealedBid is BounceBase
```

BounceDutchAuction, BounceFixedSwap, and BounceSealedBid all inherit BounceBase as their parent contract. According to the OpenZeppelin [document](#), the parent upgradeable contract should add a storage gap to avoid storage slot collisions in future upgrades. For upgradeable contracts, there must be a storage gap to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments" (quote OpenZeppelin). Otherwise it may lose the flexibility of adding new variables for the new implementation. Without a storage gap, the variable in the child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences to the child contracts, potentially causing loss of user funds or causing the contract to malfunction completely.

**Recommendation**

Recommend adding appropriate storage gaps at the end of upgradeable contracts. Please reference the OpenZeppelin upgradeable contract [document](#).

**Status**

This issue has been resolved by the team adding appropriate storage gaps at the end of BounceBase contracts.
2. Message call with hardcoded gas amount

Severity: Low  
Category: Business Logic

Target:
- contracts/BounceDutchAuction.sol
- contracts/BounceFixedSwap.sol
- contracts/BounceSealedBid.sol

Description

contracts/BounceDutchAuction.sol:L183
`payable(pool.creator).transfer(actualAmount1);`

contracts/BounceDutchAuction.sol:L227
`payable(msg.sender).transfer(unfilledAmount1);`

contracts/BounceDutchAuction.sol:L258
`payable(sender).transfer(_excessAmount1);`

contracts/BounceFixedSwap.sol:L187
`payable(msg.sender).transfer(excessAmount1);`

contracts/BounceFixedSwap.sol:L209
`payable(pool.creator).transfer(_amount1);`

contracts/BounceFixedSwap.sol:L266
`payable(msg.sender).transfer(amount1);`

contracts/BounceSealedBid.sol:L211
`payable(msg.sender).transfer(unFilledAmount1);`

contracts/BounceSealedBid.sol:L229
`payable(msg.sender).transfer(amount1);`

contracts/BounceSealedBid.sol:L259
`payable(pool.creator).transfer(actualAmount1);`

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against
reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example, EIP-1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

**Recommendation**
Avoid the use of transfer() and send() and do not otherwise specify a fixed amount of gas when performing calls. Use address.call({value: amount}('') instead. Use the checks-effects-interactions pattern and/or reentrancy locks to prevent reentrancy attacks.

**Status**
This issue has been resolved by the team using OpenZeppelin’s AddressUpgradeable.sendValue and address.call({value: amount}('') instead of transfer().
3. Lack of protection against signature replay attacks

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Category: Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td></td>
</tr>
<tr>
<td>- contracts/BounceBase.sol</td>
<td></td>
</tr>
<tr>
<td>- contracts/BounceDutchAuction.sol</td>
<td></td>
</tr>
<tr>
<td>- contracts/BounceFixedSwap.sol</td>
<td></td>
</tr>
<tr>
<td>- contracts/BounceSealedBid.sol</td>
<td></td>
</tr>
</tbody>
</table>

Description

contracts/BounceBase.sol:L58-63

```solidity
function checkCreator(bytes32 hash, uint256 expireAt, bytes memory signature) internal view {
    require(block.timestamp < expireAt, "signature expired");
    bytes32 message = keccak256(abi.encode(msg.sender, hash, block.chainid, expireAt));
    bytes32 hashMessage = message.toEthSignedMessageHash();
    require(signer == hashMessage.recover(signature), "invalid signature");
}
```

contracts/BounceDutchAuction.sol:L113

```solidity
checkCreator(keccak256(abi.encode(poolReq, PoolType.DutchAuction)), expireAt, signature);
```

contracts/BounceFixedSwap.sol:L102

```solidity
checkCreator(keccak256(abi.encode(poolReq, PoolType.FixedSwap)), expireAt, signature);
```

contracts/BounceSealedBid.sol:L94

```solidity
checkCreator(keccak256(abi.encode(poolReq, PoolType.FixedSwap)), expireAt, signature);
```

The `checkCreator()` function is used in the project to verify the signature. The function parameters are hash, expireAt and signature. The user cannot complete the create operation until the signature has been verified.

There is an expireAt parameter in the `checkCreator()` function to ensure the validity of the signature, but it lacks the functionality to prevent signature replay. A user who has obtained a valid signature can repeatedly use the same signature to call the `create()` function to create a large number of pools before the signature expires.

Recommendation

Add the parameter nonce to the `checkCreator()` function to prevent signature replay.

Status

This issue has been resolved by the team using a state variable poolMessages to record whether a signature was used.
4. Unnecessary comment of the priceHash implementation

Severity: Low
Category: Redundancy
Target:
- contracts/BounceSealedBid.sol

Description

contracts/BounceSealedBid.sol:L119-152

```solidity
function bid(
    // pool index
    uint256 index,
    // amount of token1
    uint256 amount1,
    // priceHash = keccak256(abi.encode(index, sender, amount0, amount1))
    bytes32 priceHash,
    // signMessage = keccak256(abi.encode(chainId, sender, priceHash))
    bytes memory signature,
    bytes32[] memory proof
) external payable nonReentrant isPoolExist(index) isPoolNotClosed(index) {
    checkWhitelist(index, proof);
    Pool memory pool = pools[index];
    require(pool.openAt <= block.timestamp, "pool not open");
    require(amount1 != 0, "amount1 is zero");
    require(myAmountBid1[msg.sender][index] == 0, "already bid by sender");
    bytes32 signMessage = keccak256(abi.encode(block.chainid, msg.sender, priceHash));
    bytes32 hashMessage = signMessage.toEthSignedMessageHash();
    require(signer == hashMessage.recover(signature), "invalid signature");
    address token1 = pool.token1;
    if (token1 == address(0)) {
        require(amount1 == msg.value, "invalid ETH amount");
    } else {
        IERC20Upgradeable(token1).safeTransferFrom(msg.sender, address(this), amount1);
    }
    totalBidAmount1[index] = totalBidAmount1[index].add(amount1);
    myAmountBid1[msg.sender][index] = amount1;
    myPriceHash[msg.sender][index] = priceHash;
    emit Bid(index, msg.sender, amount1, priceHash);
}
```

As the contract name BounceSealedBid implies, BounceSealedBid.sol hopes to complete the sealed bid through the `bid()` function. Each user's bid is not revealed to the public until the transaction is completed. The implementation method of the bid function is to convert the
amount1 given by the user and the desired amount0 via \texttt{keccak256(abi.encode(index, sender, amount0, amount1))} into \texttt{priceHash} and pass it as a parameter to avoid exposing one’s own bid directly.

However, there is an issue with this implementation, if the smart contract is open sourced, based on the implementation comment of \texttt{priceHash} and the index, \texttt{msg.sender}, and amount1 parameters in the emitted event of a user’s bid(), an attacker can solve the amount0 and amount1 of the user’s bid through exhaustion over a certain price range.

Therefore, the user’s bid in \texttt{BounceSealedBid} is not actually sealed.

**Recommendation**
Consider removing the \texttt{priceHash} implementation details comment.

**Status**
This issue has been resolved by adding parameter salt to prevent \texttt{priceHash} collision.
5. Single-step ownership transfer pattern risk

Severity: Low
Category: Authentication

Target:
- contracts/BounceBase.sol
- contracts/Random.sol

Description

contracts/BounceBase.sol:L5
```
import
"@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

contracts/Random.sol:L8
```
import
"@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

Inheriting from OpenZeppelin's OwnableUpgradeable contract means that you are using a single-step ownership transfer pattern. If an admin provides an incorrect address for the new owner this will result in none of the methods modified by onlyOwner being callable. The better way is to use a two-step ownership transfer pattern, where the new owner should first claim the ownership before it is transferred. There is an OpenZeppelin Ownable2StepUpgradeable contract designed for two-step ownership transferring.

**Recommendation**
Use OpenZeppelin's Ownable2StepUpgradeable.sol instead of OwnableUpgradeable.sol

**Status**
This issue has been acknowledged by the team.
6. Missing check in initialization function

Severity: Low
Category: Data Validation
Target:
- contracts/BounceBase.sol

Description
contracts/BounceBase.sol:L35-L42

```solidity
function __BounceBase_init(uint256 _txFeeRatio, address _stakeContract, address _signer)
internal onlyInitializing {
    super.__Ownable_init();
    super.__ReentrancyGuard_init();

    txFeeRatio = _txFeeRatio;
    stakeContract = _stakeContract;
    signer = _signer;
}
```

Checking addresses against zero-address during initialization or during setting is a security best-practice. However, such checks are missing for address variables during initialization. Also missing checks for txFeeRatio can lead to excessively high rates.

Recommendation
Add zero-address checks for all initializations/setters of all address state variables. Add a check for txFeeRatio.

Status
The team fixed this by adding a zero address check and txFeeRatio check during initialization.
## 2.3 Informational Findings

### 7. Floating compiler version

<table>
<thead>
<tr>
<th>Severity: Informational</th>
<th>Category: Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>all</td>
</tr>
</tbody>
</table>

**Description**

```solidity
pragma solidity ^0.8.0;
```

The BOUNCE 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 resolved by the team using locked Solidity version 0.8.17.
# 8. Redundant code

<table>
<thead>
<tr>
<th>Severity: Informational</th>
<th>Category: Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>contracts/BounceDutchAuction.sol</td>
</tr>
<tr>
<td></td>
<td>contracts/BounceLottery.sol</td>
</tr>
<tr>
<td></td>
<td>contracts/BounceSealedBid.sol</td>
</tr>
</tbody>
</table>

## Description

**contracts/BounceDutchAuction.sol:L66**

```solidity
mapping(uint256 => uint256) public amountSwap1;
```

**contracts/BounceDutchAuction.sol:L253**

```solidity
amountSwap1[index] = amountSwap1[index].add(_amount1);
```

The state variable `amountSwap1` in BounceDutchAuction is not used.

**contracts/BounceDutchAuction.sol:L106**

```solidity
poolReq.openAt < poolReq.closeAt &&
uint256(poolReq.closeAt).sub(poolReq.openAt) < 7 days,
```

**contracts/BounceSealedBid.sol:L88**

```solidity
poolReq.openAt < poolReq.closeAt &&
uint256(poolReq.closeAt).sub(poolReq.openAt) < 7 days,
```

For this `&&` condition statement, if the latter condition is true, then the former condition must also be true. Thus, the former condition check is redundant.

**contracts/BounceLottery.sol:L82**

```solidity
require(poolReq.maxPlayer < 65536, "max player must less 65536");
```

The declaration statement of the `maxPlayer` variable is `uint16 maxPlayer`; so `maxPlayer` is always less than 65536.

## Recommendation

Remove redundant code.

## Status

```
poolReq.openAt < poolReq.closeAt and require(poolReq.maxPlayer < 65536, "max player must less 65536"); has been deleted.
```
9. SafeMath library not needed since Solidity 0.8.0

Severity: Informational  
Category: Redundancy

Target:
- contracts/BounceBase.sol
- contracts/BounceDutchAuction.sol
- contracts/BounceFixedSwap.sol
- contracts/BounceSealedBid.sol
- contracts/BounceLottery.sol

Description

contracts/BounceBase.sol:L6

import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";

SafeMath is used to check underflow and overflow for arithmetic operations. However, since Solidity version 0.8.0, arithmetic operations revert on underflow and overflow by default. Since the bounce project uses a Solidity version no less than 0.8.0, it is unnecessary to use the SafeMath library.

Recommendation
Remove the SafeMath library.

Status
This issue has been acknowledged by the team.
10. Lack of NatSpec documentation

<table>
<thead>
<tr>
<th>Severity: Informational</th>
<th>Category: Code Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target:</td>
<td>All</td>
</tr>
</tbody>
</table>

**Description**

NatSpec documentation for all public methods and variables is essential for better understanding of the code by developers and auditors, and is highly recommended.

**Recommendation**

Add NatSpec documentation.

**Status**

This issue has been acknowledged by the team.
11. External call to an out of scope address

Severity: Informational  
Category: Undefined Behavior

Target:
- contracts/BounceDutchAuction.sol
- contracts/BounceFixedSwap.sol
- contracts/BounceSealedBid.sol
- contracts/BounceLottery.sol

Description

contracts/BounceDutchAuction.sol:L188

(bool success, ) = stakeContract.call{value: txFee}(abi.encodeWithSignature("depositReward()"));

contracts/BounceFixedSwap.sol:L219

(bool success, ) = stakeContract.call{value: txFee}(abi.encodeWithSignature("depositReward()"));

contracts/BounceLottery.sol:L177

(bool success, ) = stakeContract.call{value: txFee}(abi.encodeWithSignature("depositReward()"));

contracts/BounceSealedBid.sol:L264

(bool success, ) = stakeContract.call{value: txFee}(abi.encodeWithSignature("depositReward()"));

There is a Charge function in the program, which invokes the depositReward() function of stakeContract. However, the code in the stakeContract address is outside the scope of this audit and unknown to the auditors. Therefore, there may be potential risks when making an external call to stakeContract

Recommendation

Add stakeContract to the audit scope.

Status

This issue has been acknowledged by the team.
Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>SHA-1 hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>BounceBase.sol</td>
<td>474200fb15e61322dd9fb7695959df9538a78c84</td>
</tr>
<tr>
<td>BounceDutchAuction.sol</td>
<td>43a68a6008888dfed981a896efb03d5e8e31ec984</td>
</tr>
<tr>
<td>BounceFixedSwap.sol</td>
<td>84b04231342122654c1a294a11608e5e84f5c960</td>
</tr>
<tr>
<td>BounceLottery.sol</td>
<td>c25131a22f3588fa78d8d3eb4a4cc279ca699705</td>
</tr>
<tr>
<td>BounceSealedBid.sol</td>
<td>3a2c25b1defff370a9c792e8795001ad677d0615</td>
</tr>
<tr>
<td>Random.sol</td>
<td>e35a435df39910ce3f51687c6de73f5cbc358830</td>
</tr>
</tbody>
</table>