SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

ASPECTA

Overview

Project Summary

- Name: Aspecta
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/aspecta-ai/contract-v1-core
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Aspecta
Version	v2
Type	Solidity
Dates	May 30 2025
Logs	May 29 2025, May 30 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	6
Total Low-Severity issues	1
Total informational issues	2
Total	10

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	Users may fail to claim keys because of round down	High	Business Logic	Resolved
2	Launch results can be manipulated	Medium	Business Logic	Resolved
3	Improper minPrice check	Medium	Business Logic	Resolved
4	Random seed can be manipulated	Medium	Business Logic	Acknowledged
5	Suggest adding price protection for initial trade users	Medium	Business Logic	Acknowledged
6	The raffle mechanism has a flaw	Medium	Business Logic	Acknowledged
7	Centralization risk	Medium	Centralization	Acknowledged
8	Suggest adding storage gap in BondingCurve	Low	Business Logic	Acknowledged
9	Missing sanity check in setFairLaunchKeyPrice	Informational	Business Logic	Acknowledged
10	Inconsistent gaps	Informational	Business Logic	Acknowledged

2.2 Notable Findings

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

1. Users may fail to claim keys because of round down

Severity: High	Category: Business Logic
Target: - contracts/AspectaKeyPool/AspectaKeyPool.sol	

Description

In the fair launch phase, users can deposit funds via the function `fairlaunchDeposit`. After the launch deposit phase, users can claim their keys. If the fair launch sold amount is larger than our target amount, we will distribute the target keys according to different formulas.

If we set minimumClaimAmount to zero, the key amounts will be distributed according to the ratio.

Malicious users can manipulate lots of accounts to increase the `fairlaunchSoldSupply` to one very high value. This will cause all users to fail to claim a fair launch key because of the round down.

contracts/AspectakeyPool/AspectaKeyPool.sol: L593-L597

```
function _calculate_claimable_amount(
    address user,
    uint256 groupId
)
    keyAmount =
        (group.userFairlaunchData[user].keyAmount * group.targetTotalSupply) /
        group.fairlaunchSoldSupply;
}
```

Recommendation

Add one limitation for `fairlaunchSoldSupply`.

Status

This issue has been resolved by the team with commit 0e9ce45.

2. Launch result can be manipulated

Severity: Medium	Category: Business Logic

Target:
- contracts/AspectaKeyPool/AspectaKeyPool.sol

Description

In the fair launch phase, users can deposit funds via the function `fairlaunchDeposit`. After the launch deposit phase, users can claim their keys. If the fair launch sold amount is larger than our target amount, we will distribute the target keys according to different formulas.

If we set minimumClaimAmount, the key amounts will be distributed according to the raffle. The problem here is that when block.timestamp equals `fairlaunchEndTime`, we allow the fairLaunch deposit and fairLaunch claim at the same time. This will cause users to calculate the raffle with a different `randomSeed`.
contracts/AspectakeyPool/AspectaKeyPool.sol: L166-L181

```
function _checkFairlaunchStarted() internal view {
    if (
        block.timestamp < fairlaunchStartTime ||
        block.timestamp > fairlaunchEndTime ||
        fairlaunchStartTime == 0 ||
        fairlaunchEndTime == 0
    ) {
        revert FairlaunchPassedOrNotReached();
    }
}
function _checkFairlaunchEnded() internal view {
    if (block.timestamp < fairlaunchEndTime) {
        revert FairlaunchNotEnded();
    }
}
```

contracts/AspectakeyPool/AspectaKeyPool.sol: L604-L606

```
function _calculate_claimable_amount(
    address user,
    uint256 groupId
) {
    uint256 raffle = uint256(
        keccak256(abi.encodePacked(randomSeed, user, groupId))
    );
    if (raffle % group.fairlaunchSoldSupply < raffleEquivalentSupply) {
        ...
    }
}
```

Recommendation

Make sure that users cannot deposit and claim the fair launch at the same block.

Status

This issue has been resolved by the team with commit 0e9ce45.

3. Improper minPrice check

Severity: Medium	Category: Business Logic
Target: - contracts/AspectaKeyPool/AspectaKeyPool.sol	

Description

In the trade phase, the key's price will change with the curve. In order to protect users, we support the slippage control in function `sell`. The sellers can assign the minPrice that they receive to prevent them from selling keys with one cheap price.

The problem here is that sellers' actual received amount may be less than the `minPrice`.

contracts/AspectakeyPool/AspectaKeyPool.sol: L166-L181

```
function _sell(
    address seller,
    uint256 amount, // key amount to sell.
    uint256 minPrice
) internal nonReentrant {
    totalPrice = getSellPrice(amount);
    if (totalPrice < minPrice) {
        revert MinPriceNotMet(minPrice, totalPrice);
    }
    _sendFunds(payable(seller), totalPrice - protocolFee);
}
```

Recommendation

Check the `minPrice` with the actual amount that users can receive.

Status

This issue has been resolved by the team with commit 0e9ce45.

4. Random seed can be manipulated

Severity: Medium	Category: Business Logic
Target: - contracts/AspectaKeyPool/AspectaKeyPool.sol	

Description

In the fair launch phase, users can deposit funds via the function `fairlaunchDeposit`. After the launch deposit phase, users can claim their keys. If the fair launch sold amount is larger than our target amount, we will distribute the target keys according to different formulas.

If we set minimumClaimAmount, users need to win the expected key via checking the raffle.

The raffle value is related to `randomSeed`. And the `randomSeed` can be manipulated. Malicious users can try to deposit fair launch at the end of fair launch period. Users can make sure that they will win their keys via manipulating the randomSeed by modifying the `msg.value`.

contracts/AspectakeyPool/AspectaKeyPool.sol: L544-L549

```
function fairlaunchDeposit(
    uint256 groupId,
    EIP712Signature calldata _signature
) external payable whenNotPaused {
    randomSeed = uint256(
        keccak256(
            abi.encodePacked(randomSeed, msg.sender, groupId, msg.value)
        )
    );
}
```

contracts/AspectakeyPool/AspectaKeyPool.sol: L604-L609

```
function _calculate_claimable_amount(
    address user,
    uint256 groupId
) {
    uint256 raffle = uint256(
       keccak256(abi.encodePacked(randomSeed, user, groupId))
    );
    if (raffle % group.fairlaunchSoldSupply < raffleEquivalentSupply) {
       ...
    }
}
```

Recommendation

Fetch the random value from Chainlink VRF.

Status

This issue has been acknowledged by the team.

5. Suggest adding price protection for initial trade users

Severity: Medium	Category: Business Logic

Target:
- contracts/AspectaKeyPool/AspectaKeyPool.sol

Description

Before the trade begins, a `fairlaunch` will take place. The `fairlaunch` price may be significantly lower than the initial trading price (for example, in the test configuration, the `fairlaunch` price is only half of the initial price).

Once trading begins and a trader buys in, fairlaunch participants can immediately sell the same amount. They will profit by selling at the trading price. At this point, if the trader wants to sell, they must do so at the `fairlaunch` price, which would result in a significant loss.

contracts/AspectakeyPool/AspectaKeyPool.sol: L761-L725

```
function getSellPrice(
    uint256 amount
) public view returns (uint256 totalPrice) {
    if (amount < _bdcTotalSupply) {
        totalPrice = _calculateBdcSellPrice(amount);
    } else {
        totalPrice = _calculateBdcSellPrice(_bdcTotalSupply);
        totalPrice += fairlaunchKeyPrice * (amount - _bdcTotalSupply);
    }
}
```

Recommendation

It is recommended to implement price protection for traders at the early stage of trading. It can be implemented by setting a locking period for fairlaunch participants to restrict actions such as selling.

Status

This issue has been acknowledged by the team.

6. The raffle mechanism has a flaw

Severity: Medium	Category: Business Logic

Target:
- contracts/AspectaKeyPool/AspectaKeyPool.sol

Description

The current raffle system can only adjust the winning probability based on configuration, but it cannot control the actual number of winners.

As a result, it's impossible to guarantee that the `totalSupply` claimed by winners exactly reaches the `targetSupply`.

This leads to two potential issues:

1. Too many winners:

In this case, users who claim earlier can successfully purchase. However, once the total claimed amount reaches `targetSupply`, users who claim later - even if they meet the winning criteria - may not be able to purchase any amount.

This causes the claim order to affect the result, compromising fairness.

2. Too few winners:

If there are many retail users participating, it's possible that even after all winners have claimed, the total claimed amount is less than the `targetSupply`, resulting in under-allocation.

contracts/AspectakeyPool/AspectaKeyPool.sol: L550-L566

```
function _calculate_claimable_amount(
    address user,
    uint256 groupId
)
    internal
    view
    returns (
        uint256 keyAmount,
        uint256 totalPrice,
        uint256 protocolFee,
        uint256 refund
    )
{
    ...
        uint256 raffle = uint256(
            keccak256(abi.encodePacked(randomSeed, user, groupId))
        );
        if (raffle % group.fairlaunchSoldSupply < raffleEquivalentSupply) {
            keyAmount =
                (group.minimumClaimAmount *
                    group.userFairlaunchData[user].keyAmount *
```

```
                    fairlaunchKeyPriceWithFee) /
                group.maxUserDeposit;
            if (keyAmount > group.targetTotalSupply - group.totalSupply) {
                keyAmount = group.targetTotalSupply - group.totalSupply;
            }
        } else {
            keyAmount = 0;
        }
    }
    ...
}
```

Recommendation

Solving this issue requires refactoring the raffle system.

Status

This issue has been acknowledged by the team.

7. Centralization risk

Severity: Medium	Category: Business Logic
Target: - contracts/AspectaKeyPool/AspectaKeyPool.sol	

Description

In the `AspectaKeyPool`contract, there exists a privileged role called `ASPECTA_ADMIN_ROLE`. The `ASPECTA_ADMIN_ROLE` has the authority to execute some key functions such as `setFairLaunchUsersGroup`, `setTradeStartTime` and `pause`, etc.

If the `ASPECTA_ADMIN_ROLE`'s private key is compromised, an attacker could trigger these functions to manipulate the launch results.

contracts/AspectakeyPool/AspectaKeyPool.sol: L284-L296

```
function setFairlaunchUserGroup(
    uint256[] calldata _groupIds,
    uint256[] calldata _maxUserDeposits,
    uint256[] calldata _targetTotalSupplies,
    uint256[] calldata _minimumClaimAmounts
) external onlyRole(ASPECTA_ADMIN_ROLE) {
    _setFairlaunchUserGroup(
        _groupIds,
        _maxUserDeposits,
        _targetTotalSupplies,
        _minimumClaimAmounts
    );
}
```

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.

8. Suggest adding storage gap in BondingCurve

Severity: Low	Category: Business Logic

Target:
- contracts/AspectaKeyPool/AspectaKeyPool.sol

Description

Contract AspectaKeyPool inherits from contract AspectaKeyPoolFactoryStorageV1 and contract AspectaKeyPoolFactoryStorageV1 inherits from contract bonging curve.

According to the solidity's storage layout rules, the bonding curve's layout will be ordered first. The problem here is that we don't have any gap in the bonding curve. If we need to upgrade the bonding curve part and change the storage, other storage slots may be mapped incorrectly.

contracts/AspectakeyPool/AspectaKeyPool.sol: L18-L20

```
contract AspectaKeyPool is AspectaKeyPoolStorageV1 {
    ...
}
abstract contract AspectaKeyPoolStorageV1 is
   IAspectaKeyPool,
   BondingCurve,
   PausableUpgradeable,
   AccessControlUpgradeable,
   ReentrancyGuardUpgradeable,
   EIP712Upgradeable
{
   bytes32 public constant ASPECTA_ADMIN_ROLE =
       keccak256("ASPECTA_ADMIN_ROLE");

   // ---------- For Business Logic ----------
   mapping(address user => uint256) _balances;
   ...
}
```

contracts/BondingCurve/BondingCurve.sol: L7-L10

```
contract BondingCurve {
   Piecewise[] internal piecewises;
   uint256 internal _bdcTotalSupply;
}
```

Recommendation

Add one storage slot gap in the bonding curve contract.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

9. Missing sanity check in setFairLaunchKeyPrice

Severity: Informational	Category: Business Logic
Target: - contracts/AspectaKeyPool/AspectaKeyPool.sol	

Description

In AspectaKeyPool contract, the admin role is allowed to set the fair launch price anytime. If the launch price is updated after we start the fair launch. It may cause users' `depositAmount` will be less than `totalPrice`. This will cause users to fail to claim their expected keys.

contracts/AspectakeyPool/AspectaKeyPool.sol: L284-L296

```
function setFairlaunchKeyPrice(
    uint256 _fairlaunchKeyPrice
) external onlyRole(ASPECTA_ADMIN_ROLE) {
    fairlaunchKeyPrice = _fairlaunchKeyPrice;
    fairlaunchKeyPriceWithFee =
        _fairlaunchKeyPrice +
        (_fairlaunchKeyPrice * protocolFeePercentage) /
        1 ether;
}
```

contracts/AspectakeyPool/AspectaKeyPool.sol: L553-L640

```
function _calculate_claimable_amount(
    address user,
    uint256 groupId
)
    Internal view returns (
    uint256 keyAmount,
    uint256 totalPrice,
    uint256 protocolFee,
    uint256 refund
    ){
    refund =
       group.userFairlaunchData[user].depositAmount -
       totalPrice -
       protocolFee;
}
```

Recommendation

Add one sanity check to make sure that we cannot update the launch price once we start the fair launch.

Status

This issue has been acknowledged by the team.

10. Inconsistent gaps

Severity: Informational	Category: Inconsistency

Target:
- contracts/AspectaKeyPool/AspectaKeyPoolStorage.sol
- contracts/AspectaKeyPoolFactory/AspectaKeyPoolFactoryStorage.sol

Description

The protocol applies inconsistent gaps across different contracts.

This may cause inconvenience in storage management.When upgrading the contract and adding new variables, special attention must be paid to whether the changes in `__gap` between the old and new versions of the contract are correct.

contracts/AspectaKeyPoolFactory/AspectaKeyPoolFactoryStorage.sol: L39-L41

```
abstract contract AspectaKeyPoolFactoryStorageV1 is
    IAspectaKeyPoolFactory,
    UUPSUpgradeable,
    OwnableUpgradeable,
    AccessControlUpgradeable
{
    ...
    uint256[50] private __gap;
}
```

contracts/AspectaKeyPool/AspectaKeyPoolStorage.soll: L59-L60

```
abstract contract AspectaKeyPoolStorageV1 is
    IAspectaKeyPool,
    BondingCurve,
    PausableUpgradeable,
    AccessControlUpgradeable,
    ReentrancyGuardUpgradeable,
    EIP712Upgradeable
{
    ...

    uint256[49] private __gap;
}
```

Recommendation

It is recommended to revise the storage layout and reserve a consistent number of gaps.

For example, if a standard gap size of 50 slots is defined, and the `AspectaKeyPoolFactoryStorage` contract has already used 5 slots, then the length of the `__gap` array should be 45.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit a35618c:

File	SHA-1 hash
AspectaKeyPoolFactory.sol	328b9ab8cf9ca76a3147c5e89790e682d71fba06
AspectaKeyPoolFactoryStorage.sol	3723dc546a0a5e64eb0164cbc947496af7ff67e5
AspectaAssetStorage.sol	eed8b1a1fd8efe05ed210b91b3824046d6c82280
AspectaAsset.sol	a62e136f0f1050e860593f9a5403023b366ee30c
BondingCurve.sol	63c5be87357f3c3763711e89f2ba047b9b497ef6
Piecewise.sol	98f27d2925469ca8f86b70735d0b4d3afd5e5655
AspectaKeyPoolStorage.sol	4fcc0419842e0eaf2c3d4958c027b9f4c85b2409
AspectaKeyPool.sol	8ac9447f2e708562574873f8080029c2f430afb5