SALUS SECURITY

JUN 2025

CODE SECURITY ASSESSMENT

SINLAB

Overview

Project Summary

- Name: Sinlab - BAB contract
- Platform: The BSC Blockchain
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Sinlab - BAB contract
Version	v2
Type	Solidity
Dates	Jun 05 2025
Logs	May 31 2025; Jun 05 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	4
Total Low-Severity issues	4
Total informational issues	4
Total	12

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	Symbol-Uniqueness check can be bypassed with homoglyphs and invisible unicode characters	Medium	Data Validation	Resolved
2	Owner-fee underflow in createToken() blocks owner free deployment	Medium	Business logic	Resolved
3	TransferOwner always reverts due to msg.sender changed	Medium	Business logic	Resolved
4	Unintended BNB forwarded to BAB constructor is locked	Medium	Business logic	Resolved
5	Fake validator will bypass BABTHold mechanism	Low	Business logic	Acknowledged
6	Centralization risk	Low	Centralization	Acknowledged
7	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Risky External Calls	Resolved
8	Missing events for functions that change critical state	Low	Logging	Acknowledged
9	Use of floating pragma	Informational	Configuration	Resolved
10	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved
11	Incorrect deadline in exactInputSingle functions	Informational	Business logic	Resolved
12	Redundant Code	Informational	Redundancy	Acknowledged

2.2 Notable Findings

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

1. Symbol-Uniqueness check can be bypassed with homoglyphs and invisible unicode characters	
Severity: Medium	Category: Data Validation
Target: - contracts/BAB/validators/TokenValidator.sol - contracts/BABUSD1/validators/TokenValidator.sol	

Description

In the both `` `TokenValidator` `` contracts, `` `validate()` `` is meant to guarantee that no two BAB tokens share the same symbol.

The check is performed on `` `keccak256(bytes(toLower(symbol)))` ``, but `` `toLower()` `` only converts the ASCII range A–Z to lower-case. Characters outside that range—including full-width letters, mathematical bold letters, zero-width or regular spaces—are left untouched, yet render virtually identical in wallets, block explorers, and DEX UIs.

contracts/BAB/validators/TokenValidator.sol:L52-L64,
contracts/BABUSD1/validators/TokenValidator.sol:L52-L64

```
// Helper function to convert string to lowercase
function toLower(string memory str) internal pure returns (string memory) {
    …
      if ((uint8(bStr[i]) >= 65) && (uint8(bStr[i]) <= 90)) {
    …;
}
```

An attacker can therefore create a token whose symbol looks exactly like an existing, reputable token (e.g. USDC) while passing the uniqueness check: ＵＳＤＣ (full-width), USDC\u200B (zero-width space).

Users may confuse the counterfeit token for the real one, supply liquidity, or trade against it—leading to direct loss of funds and reputational damage to the protocol. And anyone can call `` `createToken()` `` in the `` `BABFactory` `` contract after paying `` `createTokenFee` ``, the attack is fully permissionless.

Recommendation

Strict character whitelist for symbols (e.g. [A–Z, 0–9] only).

Status

The team has resolved this issue in commit 4bf76b3.

2. Owner-fee underflow in createToken() blocks owner free deployment	
Severity: Medium	Category: Business logic
Target: - contracts/BAB/BABFactory.sol - contracts/BABUSD1/BABFactory.sol	

Description

In the `BABFactory` contract, the `createToken()` function allows the contract owner to deploy new BAB tokens without paying the creation fee.

contracts/BAB/BABFactory.sol:L28-L49,
contracts/BABUSD1/BABFactory.sol:L22-L42

```
function createToken(
    …
) external payable returns (address) {
    if (msg.sender != owner() && msg.value < config.createTokenFee) {
        revert InsufficientFee();
    }
    …
    BAB token = new BAB{salt: _salt, value: msg.value - config.createTokenFee}(
        name, symbol, tokenUri, address(this), msg.sender, validator
    );
    …
}
```

However, when `msg.sender == owner()`, the subsequent subtraction `msg.value - config.createTokenFee` will underflow (and immediately revert) if `config.createTokenFee > 0` and the owner supplies `msg.value == 0`.

This makes it impossible for the owner to take the “free” branch, effectively locking out the only free-deployment path.

Recommendation

Explicitly branch the subtraction only for non-owner calls.

Status

The team has resolved this issue in commit 4bf76b3.

3. TransferOwner always reverts due to msg.sender changed	
Severity: Medium	Category: Business logic
Target: - contracts/BAB/BABFactory.sol - contracts/BABUSD1/BABFactory.sol	

Description

By using `this.`, the call is made as an external transaction. Inside that call, `msg.sender` becomes the `BABFactory` contract itself, not the original owner, causing the `onlyOwner` guard in `transferOwnership()` to fail and revert every time. Thus, ownership can never be transferred.

contracts/BAB/BABFactory.sol:L45-L49,
contracts/BABUSD1/BABFactory.sol:L52-L54

```
function transferOwner(address to) external onlyOwner {
    this.transferOwnership(to);
}
```

Recommendation

Call `transferOwnership()`, without `this.` to ensure `msg.sender` remains the owner.

Status

The team has resolved this issue in commit 4bf76b3.

4. Unintended BNB forwarded to BAB constructor is locked

Severity: Medium	Category: Business logic
Target: - contracts/BABUSD1/BAB.sol - contracts/BABUSD1/BABFactory.sol	

Description

In the `BABFactory` contract, `createToken()` function sends the BNB to the `BAB` contract, but the `BAB` contract uses USD1 instead of BNB, consequently the BNB forwarded at creation just locks in the `BAB` contract.

contracts/BABUSD1/BABFactory.sol:L22-L42

```
function createToken(string memory name, string memory symbol, address validator, string
memory tokenUri, uint256 devBuy, bytes32 _salt) external payable returns (address) {
   if (msg.sender != owner() && msg.value < config.createTokenFee) {
     revert InsufficientFee();
   }
   ...
   BAB token = new BAB{salt: _salt, value: msg.value - config.createTokenFee}(name,
symbol, tokenUri, address(this), msg.sender, validator);

   if (devBuy > 0) {
     IERC20(config.usd1).transferFrom(msg.sender, address(this), devBuy);
     IERC20(config.usd1).approve(address(token), devBuy);
     token.buy(msg.sender,  msg.sender, 0, 0, devBuy, 0x0);
   }
   ...
}
```

Recommendation

It is recommended to refund to the `msg.sender` or `msg.value` validation like:

```
if (msg.sender != owner() && msg.value != config.createTokenFee)
```

Status

The team has resolved this issue in commit 4bf76b3.

5. Fake validator will create fake token	
Severity: Low	Category: Business logic
Target: - contracts/BAB/BABFactory.sol - contracts/BABUSD1/BABFactory.sol	

Description

In the `createToken()` method of the `BABFactory` contract, users are allowed to pass in any `validator` address and use it as the `validator` for the newly created `BABToken`.

contracts/BAB/BABFactory.sol:L23-L36,
contracts/BABUSD1/BABFactory.sol:L22-L42

```
function createToken(string memory name, string memory symbol, address validator, string
memory tokenUri, bytes32 _salt) external payable returns (address) {
  ...
  BAB token = new BAB{salt: _salt, value: msg.value - config.createTokenFee}(name,
symbol, tokenUri, address(this), msg.sender, validator);
  …
}
```

An attacker can deploy a `Fake Validator` contract whose validate method always returns false or does not perform any validation at all. In some cases, third-party DeFi aggregators will display fake tokens as real ones when the `BABTokenCreated` event emits.

Recommendation

The `BABFactory` contract should limit the address of `validator` and only allow validator contract addresses in the whitelist or approved by the platform.

Status

This issue has been acknowledged by the team.

6. Centralization risk	
Severity: Low	Category: Centralization
Target: - contracts/BAB/BABFactory.sol - contracts/BABUSD1/BABFactory.sol - contracts/BAB/validators/TokenValidator.sol - contracts/BABUSD1/validators/TokenValidator.sol	

Description

The `` `BABFactory` `` contract has privileged accounts. These privileged accounts can:
1. Changing the `` `BABFactory` `` configuration using the `` `setConfig()` ``.
2. Changing the `` `createTokenFee` ``to reduce the amount of money tokencreator spend on purchasing.
3. Transferring BNB which `` `BABFactory` `` holds.

The `` `TokenValidator` `` contract has privileged accounts. These privileged accounts change `` `existingSymbolHashes` `` mappings to change relationships using the `` `setSymbolState()` `` function.

If privileged accounts' private key or admin's is compromised, an attacker can steal all the tokens in the contract.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

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 and state that will transfer the ownership of this contract to a multi-sig wallet in the future.

7. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	
Severity: Low	Category: Risky External Calls
Target: - contracts/BAB/BAB.sol - contracts/BABUSD1/BAB.sol	

Description

Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

contracts/BAB/BAB.sol:L219-L240,
contracts/BABUSD1/BAB.sol:L217-L236

```
 function _handleUniswapSell(uint256 tokensToSell, uint256 minPayoutSize, uint160
sqrtPriceLimitX96) private returns (uint256) {
    transfer(address(this), tokensToSell);
    ...
 }
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Status

The team has resolved this issue in commit 4bf76b3.

8. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - contracts/BAB/validators/TokenValidator.sol - contracts/BAB/BABFactory.sol - contracts/BABUSD1/BABFactory.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `TokenValidator` contract, events are lacking in the privileged functions (e.g. `setCallerPermission()` and `setSymbolState()`).

In the both `BABFactory` contracts, events are lacking in the privileged functions (e.g. `setConfig()` and `transferOwner()`).

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

9. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.2;
```

The `BAB` contract uses a floating compiler version ^0.8.2.

Using a floating pragma ^0.8.19 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

The team has resolved this issue in commit 4bf76b3.

10. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/BAB/BABFactory.sol - contracts/BABUSD1/BABFactory.sol - contracts/BAB/validators/TokenValidator.sol - contracts/BABUSD1/validators/TokenValidator.sol	

Description

The `BABFactory` contracts and `TokenValidator`contract use a custom function `transferOwner()` and `transferOwnership()` which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

contracts/BAB/BABFactory.sol:L46-48,
contracts/BABUSD1/BABFactory.sol:L52-L54

```
function transferOwner(address to) external onlyOwner {
  this.transferOwnership(to);
}
```

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit 4bf76b3.

11. Incorrect deadline in exactInputSingle functions	
Severity: Informational	Category: Business logic
Target: - contracts/BAB/BAB.sol - contracts/BABUSD1/BAB.sol	

Description

In the `BAB` contract's `buy()` and `_handleUniswapSell()` functions, the deadline parameter for `exactInputSingle()` function calls is set as `MAX_UINT256`. This kind of order may be subject to MEV attack.

contracts/BAB/BAB.sol:L135-L144,
contracts/BABUSD1/BAB.sol:L129-L138

```
ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
      tokenIn: address(this),
      tokenOut: WETH,
      fee: LP_FEE,
      recipient: address(this),
      deadline: MAX_UINT256,
      amountIn: tokensToSell,
      amountOutMinimum: minPayoutSize,
      sqrtPriceLimitX96: sqrtPriceLimitX96
    });
```

Recommendation

Using `block.timestamp` as the deadline.

Status

The team has resolved this issue in commit 4bf76b3.

12. Redundant Code

Severity: Informational	Category: Redundancy

Target:
- contracts/BAB/validators/BABTValidator.sol
- contracts/BABUSD1/validators/BABTValidator.sol
- contracts/BAB/validators/TestValidator.sol
- contracts/BABUSD1/validators/TestValidator.sol
- contracts/BAB/validators/TokenValidator.sol
- contracts/BABUSD1/validators/TokenValidator.sol

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following functions are not being utilized:

contracts/BAB/validators/BABTValidator.sol:L15-L20,
contracts/BABUSD1/validators/BABTValidator.sol:L15-L20

```
function validate(address sender, uint256 value, bytes32 data) public view returns
(bool) {…}
```

contracts/BAB/validators/TestValidator.sol:L14-L20,
contracts/BABUSD1/validators/TestValidator.sol:L14-L20

```
function validate(address sender, uint256 value, bytes32 data) public view returns
(bool) {…}
```

contracts/BAB/validators/TokenValidator.sol:L19-L35,
contracts/BABUSD1/validators/TokenValidator.sol:L19-L35

```
function validate(string memory name, string memory symbol) external {…}
```

Recommendation

Consider removing the redundant code.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit b72a78f:

File	SHA-1 hash
contracts/BAB/BABFactory.sol	329dc5eb20689fb0d908e1ef03e06c7e1eb994c6
contracts/BAB/BAB.sol	ff8316e966e307c51a006c74c31dd05d4a927c90
contracts/BAB/BondingCurve.sol	be366f4bd18326a99be3dea92aa3c6b60b69d1c7
contracts/BAB/validators/BABTValidator.sol	0d1850b361cdac6e88a1e74b78161e6c2aad16e3
contracts/BAB/validators/TokenValidator.sol	2ffc26520caa0a58af9ce81e6a9ec68b6c6a0276
contracts/BAB/validators/TestValidator.sol	04ceea561f0bf12ab98dcd3b2666b5b84d088361
contracts/BABUSD1/BABFactory.sol	83e6a958cda3339bdda3bf5b5d398fab108d586f
contracts/BABUSD1/BAB.sol	c50df96b9739b78f958d3592032dcf11adc4e77d
contracts/BABUSD1/BondingCurve.sol	5e5fd882a511488d389392f6fdac24f41b2a9a12
contracts/BABUSD1/validators/BABTValidator.sol	0d1850b361cdac6e88a1e74b78161e6c2aad16e3
contracts/BABUSD1/validators/TokenValidator.sol	2ffc26520caa0a58af9ce81e6a9ec68b6c6a0276
contracts/BABUSD1/validators/TestValidator.sol	04ceea561f0bf12ab98dcd3b2666b5b84d088361