



# CODE SECURITY ASSESSMENT

B 4 0 2 A I

# Overview

## Project Summary

- Name: B402 AI - B402 contract
- Platform: BSC
- Language: Solidity
- Repository:
  - <https://github.com/b402-ai/b402-contracts>
- Audit Range: See [Appendix - 1](#)

## Project Dashboard

### Application Summary

Name	B402 AI - B402 contract
Version	v2
Type	Solidity
Dates	Nov 01 2025
Logs	Oct 31 2025; Nov 01 2025

### Vulnerability Summary

Vulnerability	Num of Vulnerability	Num of Resolved Vulnerability
Total High-Severity issues	0	0
Total Medium-Severity issues	0	0
Total Low-Severity issues	2	2
Total informational issues	2	1
Total	4	3

## Contact

E-mail: [support@salusec.io](mailto:support@salusec.io)

## Risk Level Description

<b>High Risk</b>	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.
<b>Medium Risk</b>	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.
<b>Low Risk</b>	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.
<b>Informational</b>	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

# Content

<b>Introduction</b>	<b>4</b>
1.1 About SALUS	4
1.2 Audit Breakdown	4
1.3 Disclaimer	4
<b>Findings</b>	<b>5</b>
2.1 Summary of Findings	5
2.2 Notable Findings	6
1. Signature mechanism doesn't support EIP1271	6
2. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	7
2.3 Informational Findings	8
3. Authorizationstate's return value is not clear	8
4. Mapping Declaration Lacks Named Parameters for Improved Readability	9
<b>Appendix</b>	<b>10</b>
Appendix 1 - Files in Scope	10

# 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](https://twitter.com/salus_sec)), or Email ([support@salusec.io](mailto: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	<a href="#">Signature mechanism doesn't support EIP1271</a>	Low	Cryptography	Resolved
2	<a href="#">Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()</a>	Low	Risky External Calls	Resolved
3	Authorizationstate's return value is not clear	Informational	Business Logic	Acknowledged
4	Mapping declaration lacks named parameters for improved readability	Informational	Code Quality	Resolved

## 2.2 Notable Findings

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

### 1. Signature mechanism doesn't support EIP1271

Severity: Low

Category: Cryptography

Target:

- src/core/B402RelayerV3.sol

### Description

The `transferWithAuthorization()` function allows users to submit a signature for auto-pay and the `cancelAuthorization()` function is used to cancel it. But the verification process only supports the EOAs signature.

[EIP1271](#) is a standard signature validation method for contracts. Adding this type of support can increase the number of potential users.

src/core/B402RelayerV3.sol:L195-L154

```
function transferWithAuthorization(...) external nonReentrant whenNotPaused {
    ...
    // Verify signature using EIP-712
    bytes32 structHash = keccak256(
        abi.encode(TRANSFER_WITH_AUTHORIZATION_TYPEHASH, token, from, to, value,
        validAfter, validBefore, nonce)
    );
    bytes32 digest = _hashTypedDataV4(structHash);
    address signer = ECDSA.recover(digest, v, r, s);
    if (signer != from) revert InvalidSignature(from, signer);
    ...
}
function cancelAuthorization(address authorizer, bytes32 nonce, uint8 v, bytes32 r,
bytes32 s) external {
    ...
    // Verify signature using EIP-712
    bytes32 structHash = keccak256(abi.encode(CANCEL_AUTHORIZATION_TYPEHASH, authorizer,
    nonce));

    bytes32 digest = _hashTypedDataV4(structHash);
    address signer = ECDSA.recover(digest, v, r, s);
    if (signer != authorizer) revert InvalidSignature(authorizer, signer);
    ...
}
```

### Recommendation

Using Openzeppelin's SignatureChecker library instead of the existing signature scheme.

### Status

The team has resolved this issue in commit [289d885](#).

## 2. Use `safeTransfer()/safeTransferFrom()` instead of `transfer()/transferFrom()`

Severity: Low

Category: Risky External Calls

Target:

- `src/core/B402RelayerV3.sol`

### Description

The contract calls `transferFrom()` and decodes the return value as bool:

`src/core/B402RelayerV3.sol:L134-L135`

```
bool success = tokenContract.transferFrom(from, to, value);  
if (!success) revert TransferFailed(token);
```

In practice, many widely used tokens are non-standard: some do not return any value, while others return non-boolean data or use false instead of reverting to signal failure. For tokens that do not return data, ABI decoding to bool will revert, causing a compatibility DoS.

### 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 [fc3c45c](#).

## 2.3 Informational Findings

### 3. Authorizationstate's return value is not clear

Severity: Informational

Category: Business Logic

Target:

- src/core/B402RelayerV3.sol

### Description

The `\_authorizationStates` mapping stores a single boolean that does not distinguish between an authorization being consumed (`AuthorizationUsed`) or intentionally revoked (`AuthorizationCanceled`). Therefore, the `authorizationState()` function's return value could be misleading.

src/core/B402RelayerV3.sol:L159-L162

```
/// @inheritdoc IB402RelayerV3
function authorizationState(address authorizer, bytes32 nonce) external view returns
(bool) {
    return _authorizationStates[authorizer][nonce];
}
```

### Recommendation

Consider using distinct states (e.g., used vs. canceled).

### Status

This issue has been acknowledged by the team.

## 4. Mapping declaration lacks named parameters for improved readability

Severity: Informational

Category: Code Quality

Target:

- src/core/B402RelayerV3.sol

### Description

Since Solidity version 0.8.18, the language allows developers to add named key and value identifiers in mapping declarations using the syntax:

```
mapping(KeyType keyName => ValueType valueName)
```

This enhances code readability and clarifies how a mapping is intended to be used. In the `B402RelayerV3` contract, the mappings are declared without named parameters, making the purpose of the key and value less obvious upon inspection.

src/core/B402RelayerV3.sol:L43-L47

```
/// @dev Track used nonces
mapping(address => mapping(bytes32 => bool)) private _authorizationStates;

/// @inheritdoc IB402RelayerV3
mapping(address => bool) public whitelistedTokens;
```

### Recommendation

Consider updating the mapping declaration to explicitly name the key and value identifiers, enhancing readability and making the mapping's purpose clearer.

### Status

The team has resolved this issue in commit [b7a1808](#).

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files in commit [84a22a9](#):

File	SHA-1 hash
B402RelayerV3.sol	d650eaddc6d65ea7bb74732a8960c83bad674c17