SALUS SECURITY

SEP 2025

CODE SECURITY ASSESSMENT

BITWAY LABS

Overview

Project Summary

- Name: Bitway Token
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/bitwaylabs/evm-contracts/tree/main
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Bitway Token
Version	v2
Type	Solidity
Dates	Sep 13 2025
Logs	Sep 12 2025; Sep 13 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	5
Total	7

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	Centralization risk with initial token distribution	Low	Centralization	Acknowledged
2	Lack of testing	Low	Configuration	Acknowledged
3	Duplicate Imports	Informational	Redundancy	Acknowledged
4	Function Visibility Overly Permissive	Informational	Code Quality	Acknowledged
5	Lack of Security Contact Information for Responsible Disclosure	Informational	Configuration	Acknowledged
6	Missing zero address checks	Informational	Data Validation	Acknowledged
7	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Centralization risk with initial token distribution	
Severity: Low	Category: Centralization
Target: - contracts/BitwayToken.sol	

Description

contracts/BitwayToken.sol:L19 - L24

```
constructor(
    uint256 totalSupply,
    address treasury
) ERC20("Bitway Token", "BTW") ERC20Permit("Bitway Token") {
    _mint(treasury, totalSupply);
}
```

When the contract is deployed, `` `$BTW` `` is sent to one account. This account then has full control over the token distribution. If it is an EOA account, any compromise of its private key could drastically affect the project – for example, attackers could manipulate the price of `` `$BTW` `` on the DEX if they gain access to the private key.

Recommendation

It is recommended to transfer tokens to a multi-sig account and promote transparency by providing a breakdown of the intended initial token distribution in a public location.

Status

This issue has been acknowledged by the team.

2. Lack of testing	
Severity: Low	Category: Configuration
Target: - test/BitwayToken.js	

Description

Currently, while a test file is present, it lacks any meaningful content, and the existing functionalities have not been validated against the expected behavior. This absence of proper testing increases the risk of undiscovered defects or regressions, which in turn may compromise the overall security, reliability, and maintainability of the contract. Comprehensive and well-structured test coverage is strongly recommended to ensure robust verification of critical functionalities.

Recommendation

It is recommended to increase the test coverage.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Duplicate Imports	
Severity: Informational	Category: Redundancy
Target: - contracts/BitwayToken.sol	

Description

In the `BitwayToken` contract, a duplicate import instance was found:

The `BitwayToken.sol` file imports `ERC20.sol` which is already imported as a result of importing `ERC20Permit.sol`.

Recommendation

It is recommended to remove duplicate import statements and explicitly import ERC20 while importing ERC20Permit to improve the overall clarity and readability of the code base.

Status

This issue has been acknowledged by the team.

4. Function Visibility Overly Permissive	
Severity: Informational	Category: Code Quality
Target: - contracts/BitwayToken.sol	

Description

The `` `burn()` `` function in the contract is declared with broader visibility than necessary (public instead of external). Overly permissive visibility can expose internal logic to unintended callers, increase the attack surface, and slightly increase gas costs. Functions intended to be called externally only do not need public visibility and could be declared external to better communicate intended usage and optimize gas.
contracts/BitwayToken.sol:L30 - L32

```
function burn(uint256 value) public {
    _burn(_msgSender(), value);
}
```

Recommendation

Consider changing the visibility of the above function.

Status

This issue has been acknowledged by the team.

5. Lack of Security Contact Information for Responsible Disclosure	
Severity: Informational	Category: Configuration
Target: - contracts/BitwayToken.sol	

Description

The contract `BitwayToken` does not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

Status

This issue has been acknowledged by the team.

6. Missing zero address checks	
Severity: Informational	Category: Data Validation
Target: - contracts/BitwayToken.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for address variables `treasury`.

contracts/BitwayToken.sol:L19 - L24

```
constructor(
    uint256 totalSupply,
    address treasury
) ERC20("Bitway Token", "BTW") ERC20Permit("Bitway Token") {
    _mint(treasury, totalSupply);
}
```

Recommendation

Consider adding zero address checks for address variables `treasury`.

Status

This issue has been acknowledged by the team.

7. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - contracts/BitwayToken.sol	

Description

```
pragma solidity ^0.8.20;
```

The `BitwayToken` uses a floating compiler version `^0.8.20`.

Using a floating pragma `^0.8.20` 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 acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 42ea743:

File	SHA-1 hash
contracts/BitwayToken.sol	8aab6792bb7e201ace2973484d53e9f56d41a413