QuillAudits

AUDIT REPORT

October 2025

For

blazpay

Table of Content

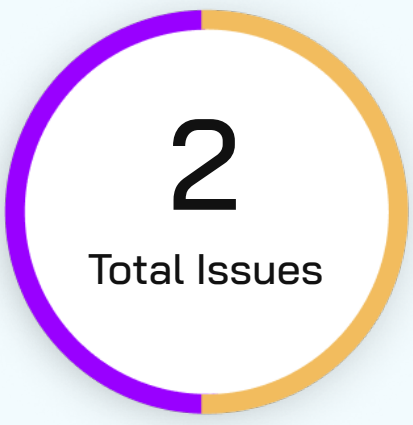
Executive Summary

Project Name	Blaz Token
Protocol Type	Token
Project URL	https://blazpay.com/
Overview	The Blazpay contract is a custom ERC20 token built on top of OpenZeppelin libraries, combining several standard extensions. It implements a fixed-supply token with additional administrative and user features.
Audit Scope	The scope of this Audit was to analyze the Blaz Token Smart Contracts for quality, security, and correctness.
Source Code Link	https://github.com/blazpay/blaz-token/blob/main/Blazpay.sol
Branch	Main
Commit Hash	403ded9cc485bf00562e934d0776160fabe08946
Contracts in Scope	Blazpay.sol
Language	Solidity
Blockchain	BnB Smart Chain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	13th October 2025
Mainnet Address	https://bscscan.com/address/0x6b1385a56f7689f6fc26c9857531e3a03a17052a

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	0 (0.0%)
High	0 (0.0%)
Medium	0 (0.0%)
Low	1 (50.0%)
Informational	1 (50.0%)

Severity

Issues	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	1	1
Partially Resolved	0	0	0	0	0
Resolved	0	0	0	0	0

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Consider using Ownable2Step instead	Low	Acknowledged
2	Lack of zero address check in constructor	Informational	Acknowledged

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls

- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw
- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Low Severity Issues

Consider using Ownable2Step instead

Acknowledged

Path

BlazPay.sol

Description

The contract currently inherits from OpenZeppelin'sOwnable,which uses a single-step ownership transfer pattern. This means that when the owner calls transferOwnership(newOwner)ownership is immediately transferred to the new address. If the new address is mis-typed or not controlled by the intended entity, ownership may be irretrievably lost.

Impact

- Ownership could be permanently lost due to accidental mis-transfer.
- Loss of administrative control over pausing/unpausing and other future privileged functions.

Likelihood

Low (requires a mistake during ownership transfer).

POC

```
// Owner accidentally calls:
contract.transferOwnership(0xDead…beef);

// Contract ownership is now irrecoverable.
```

Recommendation

Use OpenZeppelin's Ownable2Step, which requires the new owner to explicitly accept ownership via acceptOwnership(). This prevents accidental loss of control.

Informational Issues

Lack of zero address check in constructor

Acknowledged

Path

Blazpay.sol

Function Name

```
constructor(address initialOwner)
```

Description

The constructor does not explicitly check that initialOwner is not the zero address. Passing address(0) as the initial owner would cause _mint(initialOwner, TOTAL_SUPPLY) to revert, since ERC20 does not allow minting to the zero address. While this protects against minting, the revert reason is not explicit and could confuse integrators.

Impact

- No funds are at risk.
- Deployment would revert if address(0) is passed.
- The lack of explicit revert reason reduces clarity.

Likelihood

Low (only occurs if deployer mis-configures).

POC

```
// Deploying with zero owner
new Blazpay(address(0));
// Deployment reverts with a generic "ERC20: mint to the zero address"
```

Recommendation

Add an explicit address(0) check

Functional Tests

Some of the tests performed are mentioned below:

- **Should deploy successfully with valid initialOwner**
- **Should revert deployment if initialOwner is zero address**
- **Should mint total supply to initialOwner**
- **Should return correct totalSupply, balanceOf**
- **Owner should be able to pause the contract**
- **Should allow valid permit() calls (off-chain signature approval)**

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Threat Model

Contract	Function	Threats
Blazpay.sol	pause(), unpause()	**Inputs** • No inputs. Only callable by owner. **Constraints** • Contract must not already be paused. **Impact** • Freezes all transfers, mints, and burns (due to _update override). • Does not freeze approve() or permit() unless custom logic is added. **Branches and code coverage** • Should pause transfers, burn, and mint. • Should emit Paused event. • Should revert if called when already paused. **Negative Behaviour** • Malicious owner can freeze all user activity. • Unexpected: Approvals and permits can still be granted while paused.

Contract	Function	Threats
	burn(uint256 amount) / burnFrom(address account, uint256 amount)	**Inputs** • amount (controlled by caller). • account (in burnFrom, controlled by caller if allowance exists). **Constraints** • Caller must own tokens (burn). • Caller must have allowance for another's tokens (burnFrom). • Contract must not be paused (blocked in _update). **Impact** • Reduces supply permanently. • Emits Transfer(account, 0, amount). **Branches & Coverage** • Should reduce supply correctly. • Should emit event. • Should revert when paused. • Should revert if balance/ allowance insufficient. **Negative Behaviour** • If burning is needed as part of protocol logic, pausing unintentionally blocks it.

Contract	Function	Threats
	permit()	**Inputs** • owner, spender, value, deadline, v, r, s. **Constraints** • Signature must be valid. • Deadline must not be expired. • Nonce must match. **Impact** • Updates allowance off-chain (gasless approvals). • Emits Approval. **Branches & Code Coverage** • Should succeed with valid signature. • Should revert with expired deadline. • Should revert with invalid signature. • Should prevent replay (nonce). **Negative Behaviour** • Still works when contract is paused → approvals can be set even during freeze. • Replay protection relies on correct nonce handling (covered by OZ).

Note to Users/Trust Assumptions

Centralization risk

single-address control (high): Because the entire supply was minted to one recipient address at deployment, that address controls the entire circulating supply until tokens are distributed. If that address is a team, exchange, or a private wallet, they can move or sell tokens at any time. Treat tokens with a single initial holder as high-risk for sudden dumps. Verify who controls the recipient.

Permit (gasless approvals)

permit is present — convenient UX, but always confirm front-end uses it safely (signing UX is still off-chain/UX risk).

Closing Summary

In this report, we have considered the security of Blaz Token. We performed our audit according to the procedure described above.

No critical issues in Blaz Token, just 1 Low and 1 informational severity issues were found,which the Blaz Token team has acknowledged.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

October 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com