KREDICT

Smart Contract Audit

Astra Bitcoin (ABTC)

Deliverable
Smart Contract Audit Report
Security Report **Mar 2025**

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions, and recommendations in this publication are elaborated in the specific for only the project.

KREDICT does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings, and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

KREDICT retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects to protect all customer security purposes. The description containing confidential information can be used internally by the Customer or disclosed publicly after all vulnerabilities are fixed - upon the Customer's decision.

Overview

Background

Astra Bitcoin requested that KREDICT perform an Extensive Smart Contract audit of their Smart Contract.

Project Dates

The following is the project schedule for this review and report:

Mar 24, 2025

➔ Smart Contract Review Completed (Completed)

Mar 25, 2025

➔ Delivery of Smart Contract Audit Report (Completed)

Review Team

The following KREDICT team members participated in this review:

➔ **Abhishek Mishra**, Security Researcher, and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigated, and reviewed the smart contract of **Astra Bitcoin**.

The following documentation repositories were considered in-scope for the review:

➔ **Astra Bitcoin (ABTC)**

Explorer Link

0x8C45bC48a451F48cca403ff2DDba2dA0EA68c404

Token Name	Astra Bitcoin
Symbol	ABTC
Chain	Ethereum
Contract Address	0x8C45bC48a451F48cca403ff2DDba2dA0EA68c404
Supply	5,000,000
Decimal	18
Burn	0%
Proxy	No
Code Visible	Yes
Can Set Fees	Safe
Can take Back Ownership	Safe
Is Mintable	Safe
Can Pause	Yes
Can Blacklist	Safe
Can Burn	Safe

Contract Overview:

Name: UltimateTokenOwnable

Solidity Version: 0.8.28

Inheritance: `Initializable`, `ERC20TokenMetadata`, `ERC20Base`, `ERC20Capped`, `Pausable`, `Ownable`

Dependencies: OpenZeppelin contracts

Purpose: An ERC20 token with additional features such as ownership control, pausing, capping, minting, burning, and token metadata.

Findings

1. High Severity Issues

No high-severity issues were identified.

---

2. Medium Severity Issues

a. Centralized Control Risk

- **Description:** The contract relies heavily on the onlyOwner modifier for critical functions (pause, unpause, mint, setTokenURI). If the owner's private key is compromised, an attacker could:
  - Mint unlimited tokens up to _maxSupply.
  - Pause the contract indefinitely, freezing all transfers.
  - Alter the token URI, potentially misleading users.
- **Impact:** Loss of trust in the token and potential financial damage to holders.
- **Recommendation:**
  - Consider implementing multi-signature ownership (e.g., using OpenZeppelin's Ownable2Step).
  - Add time-locks or governance mechanisms to delay critical actions, giving users time to react.

---

3. Low Severity Issues

a. Missing Input Validation

- **Description:** The mint function does not explicitly check if to is the zero address (address(0)). While OpenZeppelin's _mint function handles this internally, explicit validation improves clarity and robustness.

- **Impact:** None currently (due to OpenZeppelin's safeguards), but it's a best practice to enforce explicitly.

b. Potential Integer Overflow in _maxSupply

- **Description:** Although Solidity 0.8. x includes built-in overflow checks, the contract does not explicitly document or enforce a reasonable upper bound for _maxSupply during initialization. A very large _maxSupply could lead to unexpected behavior in applications relying on the token.
- **Impact:** Minor; depends on the intended use case.
- **Recommendation:** Consider adding a practical cap (e.g., 2**128 - 1) or documenting the intended range for _maxSupply.

---

4. Gas Optimization Opportunities

a. Unnecessary Inheritance Complexity

- **Description:** The contract inherits from multiple OpenZeppelin contracts (ERC20Base, ERC20Capped, etc.), but ERC20Capped already extends ERC20Base. This redundant inheritance increases deployment gas costs slightly.
- **Impact:** Minor gas overhead during deployment.

b. String Parameters in initialize

- **Description:** _name, _symbol, and tokenUri_ are passed as string memory, which can lead to higher gas costs for long strings. Using calldata instead of memory could reduce gas usage.
- **Impact:** Minor gas savings during deployment.

---

5. Best Practice Observations

a. Token URI Mutability

- **Description:** The setTokenURI function allows the owner to change the token URI after deployment. While this is a feature, it might confuse users expecting a static URI (common in NFT standards like ERC721).
- **Impact:** Potential user confusion.
- **Recommendation:** If mutability is intentional, document it. If not, consider removing setTokenURI or making it immutable after initialization.

b. No Emergency Stop Mechanism Beyond Pause

- **Description:** The pause function stops transfers but doesn't provide a broader emergency mechanism (e.g., recovering stuck tokens or blacklisting malicious actors).
- **Impact:** Limited flexibility in emergencies.
- **Recommendation:** Evaluate if additional safety features (e.g., token recovery for accidental transfers) are needed based on the use case.

Security Assessment

Positive Aspects:

1. **Use of OpenZeppelin Libraries:** The contract leverages well-audited OpenZeppelin implementations, reducing the likelihood of common vulnerabilities (e.g., reentrancy, overflow).
2. **Initializer Pattern:** The use of Initializable ensures the contract can be deployed via a proxy, supporting upgradeability.
3. **Access Control:** The onlyOwner modifier restricts sensitive functions appropriately.
4. **Paused Transfers:** The _update override with whenNotPaused ensures transfers are halted when paused, as intended.

Potential Attack Vectors:

1. **Owner Compromise:** As noted, a compromised owner could abuse mint or pause.
2. **Denial of Service (DoS):** If the contract is paused indefinitely, users cannot transfer tokens, though this is a design choice rather than a vulnerability.
3. **Front-Running:** The initialize function could be front-run if deployed without a factory or proxy, but this is mitigated if deployed via a trusted factory (e.g., CreateMyToken).

Summary

Overall Assessment:

The UltimateTokenOwnable contract is well-structured and leverages secure OpenZeppelin components. No critical vulnerabilities were found, but there are areas for improvement in terms of decentralization (owner control), and gas efficiency.

Recommendations Recap:

1. Mitigate centralized control risks (e.g., multisig, time-locks).
2. Include input validation for clarity.
3. Optimize gas usage by refining inheritance and using calldata.
4. Document the intended behavior of mutable fields like tokenUri.

About KREDICT

We believe that people have a fundamental need for security and that the use of secure solutions enables every person to use the Internet and every other connected technology. We aim to provide security consulting services to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The KREDICT team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed cryptographic protocols and distributed system architecture implementations, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please email us at **support@kredict.com**