Comprehensive Security Audit Report

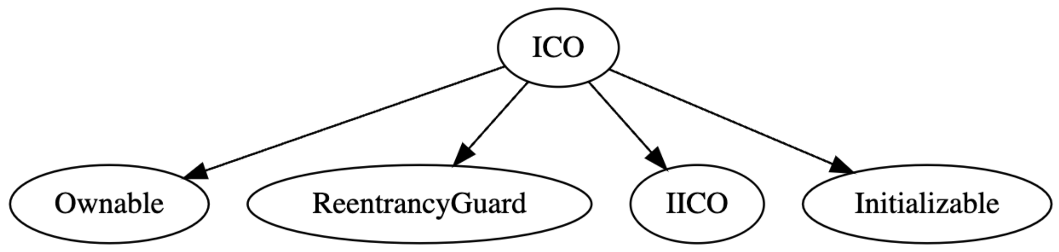
Project Name: BMIC
Audit Date: October 23, 2025
Auditor: Senior Solidity Security Auditor
Scope: BMICToken, ICO.sol, Token.sol(Solidity =0.8.28)
Codebase: Provided by file
Methodology: Line-by-line manual review, automated analysis, architectural assessment

EXECUTIVE SUMMARY

Risk Summary
• Total Contracts Analyzed: 3 (BMICToken, ICO.sol, Token.sol)
• High Vulnerabilities: 3
• Medium Vulnerabilities: 3
• Low/Informational: 2

Deployment Readiness
NOT READY FOR MAINNET - Vulnerabilities must be resolved first.

CONTRACT INHERITANCE TREE


$$\begin{tikzcd}
 & & \text{ICO} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Ownable} & \text{ReentrancyGuard} & \text{IICO} & \text{Initializable}
\end{tikzcd}$$

Testnet Version

BMIC: **https://testnet.bscscan.com/address/0x7d3f08A719148a44A950A5Ec569DFD81EDc87009#code**

ICO: **https://testnet.bscscan.com/address/0x075426Be0eBdc712076ebEe093A3989c6c29745f#code**

Token: **https://testnet.bscscan.com/address/0x910044Bae587969b8722111227dfb577e17cfF44#code**

OWNERSHIP PRIVILEGES

ICO.sol
1- The owner can enable or disable token claiming. (via enableClaimTokens())
2- The owner can set the address of the ICO token. (via setTokenAddress())
3- The owner can change the ICO token's price. (via changePrice())
4- The owner can send any remaining ICO tokens from the contract to the receiver address.
5 The owner can update the receiver address for collected funds. (via updateReceiverAddress())
6- The owner can transfer ownership of the contract to a new address. (inherited from Ownable)
7- The proxy owner can set the contract's maintenance.
8- The proxy owner can transfer ownership of the proxy.

BMICToken.sol
1- The owner can mint new tokens up to the predefined maximum supply.
2- The owner can burn tokens from their own balance.

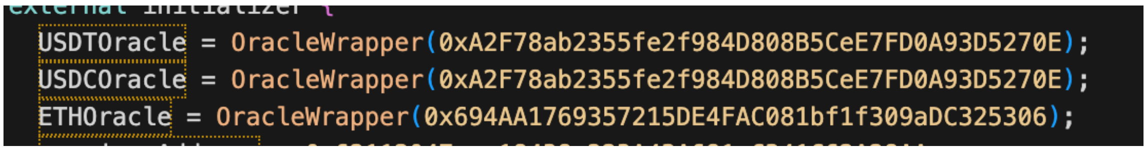
Token.sol
1- The owner can mint unlimited tokens.

DETAILED VULNERABILITY ANALYSIS

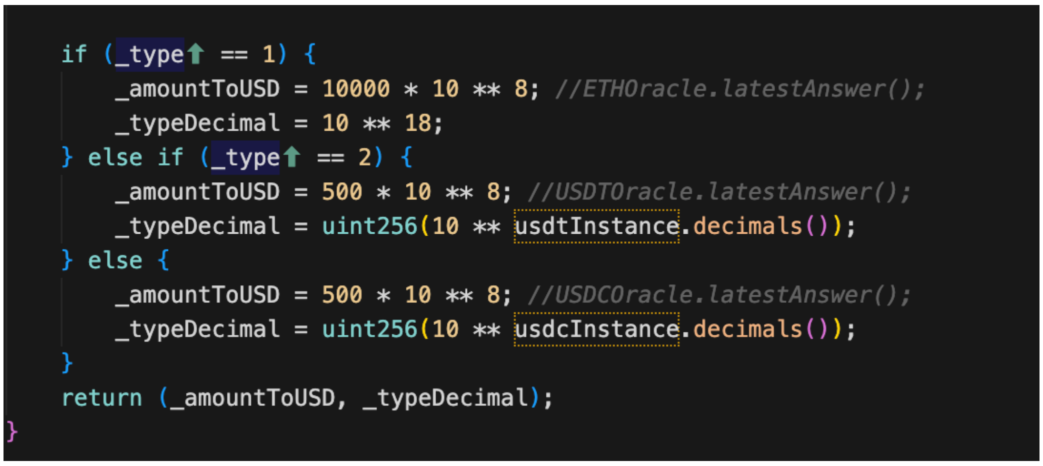
1- Title: Hardcoded & Unused Oracle Prices

Severity: High

File and Code Location: ICO.sol - Lines 52-54 (initialization) and 179, 182, 185 (usage/mocking)



```
  USDTOracle = OracleWrapper(0xA2F78ab2355fe2f984D808B5CeE7FD0A93D5270E);
  USDCOracle = OracleWrapper(0xA2F78ab2355fe2f984D808B5CeE7FD0A93D5270E);
  ETHOracle = OracleWrapper(0x694AA1769357215DE4FAC081bf1f309aDC325306);
```



```
    if (_type == 1) {
        _amountToUSD = 10000 * 10 ** 8; //ETHOracle.latestAnswer();
        _typeDecimal = 10 ** 18;
    } else if (_type == 2) {
        _amountToUSD = 500 * 10 ** 8; //USDTOracle.latestAnswer();
        _typeDecimal = uint256(10 ** usdtInstance.decimals());
    } else {
        _amountToUSD = 500 * 10 ** 8; //USDCOracle.latestAnswer();
        _typeDecimal = uint256(10 ** usdcInstance.decimals());
    }
    return (_amountToUSD, _typeDecimal);
}
```

Description: The contract sets oracle addresses in initialize but the cryptoValues function explicitly uses static, hardcoded values instead of dynamic oracle calls. This fundamentally breaks the ICO's market-based pricing mechanism, leading to incorrect token distribution and rendering the token sale non-functional in production..

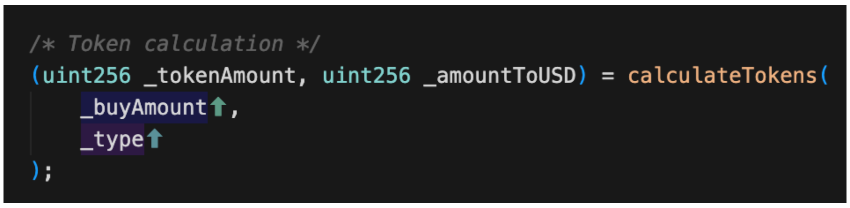
Mitigation: Replace static values in cryptoValues with actual oracle latestAnswer() calls. Parameterize oracle addresses in initialize for flexibility across networks and add on-chain validation.

Status: OPEN

2- Title: Static Pricing & Flawed Token Math

Severity: High

File and Code Location: ICO.sol - Lines 107-110 (buyTokens calling calculateTokens), and Lines 60, 152, 179, 182, 185 (upstream flaws).



```
/* Token calculation */
(uint256 _tokenAmount, uint256 _amountToUSD) = calculateTokens(
    _buyAmount,
    _type
);
```

Description: buyTokens relies on calculateTokens, which uses cryptoValues's mocked static prices (e.g., ETH at $10k, USDT/USDC at $500) and a hardcoded ICO token decimal count. This fundamentally breaks market-based pricing, guaranteeing incorrect token distribution.

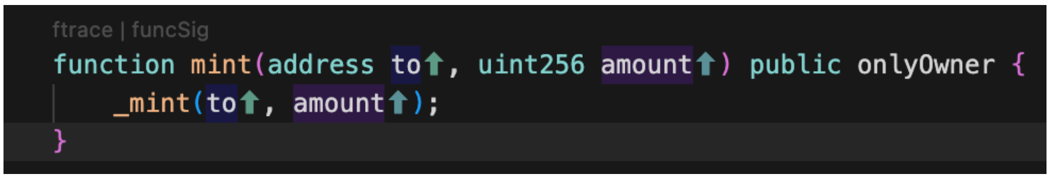
Mitigation: Implement live oracle calls in cryptoValues and dynamically fetch tokenInstance decimals. This ensures accurate, market-based pricing and correct token allocations for buyers.

Status: OPEN

3- Title: The owner can mint Unlimited Tokens

Severity: High

File and Code Location: Token.sol - Lines 21-23



```
ftrace | funcSig
function mint(address to, uint256 amount) public onlyOwner {
    _mint(to, amount);
}
```

Description: The owner is able to mint unlimited tokens, which is not recommended as this functionality can cause the token to lose its value, and the owner can also use it to manipulate the price of the token.

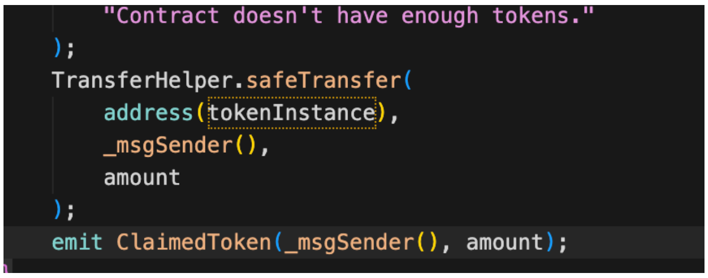
Mitigation: It is recommended that the total supply of the token should not be changed after initial deployment.

Status: OPEN

4- Title: claimTokens Distributes Incorrect Token Amounts

Severity: Medium

File and Code Location: ICO.sol - Line 222, 216



```
        "Contract doesn't have enough tokens."
    );
    TransferHelper.safeTransfer(
        address(tokenInstance),
        _msgSender(),
        amount
    );
    emit ClaimedToken(_msgSender(), amount);
```

Description: The claimTokens function transfers an amount directly derived from userMapping. This amount is catastrophically incorrect due to upstream flaws in calculateTokens, which uses mocked static oracle prices and a hardcoded ICO token decimal factor.

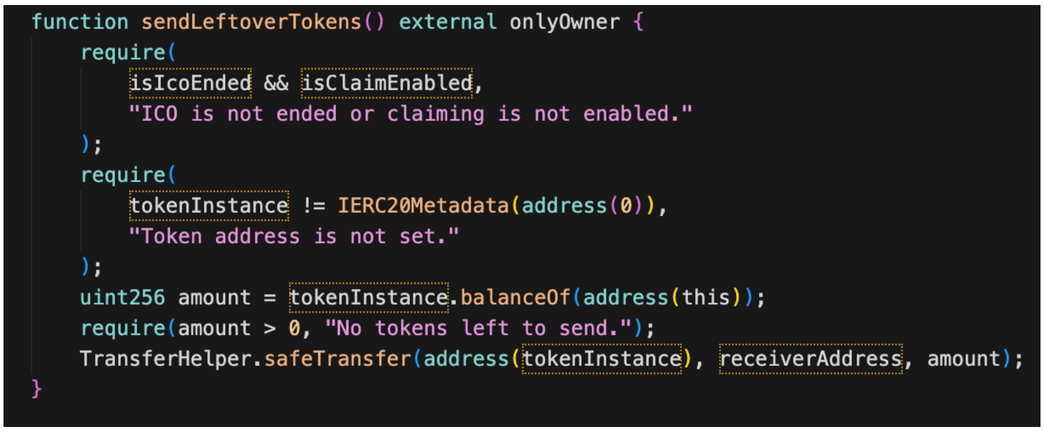
Mitigation: Resolve critical flaws in cryptoValues (live oracles, dynamic decimals) and tokenDecimal (dynamic fetch). This ensures _tokenAmount is correct, making claimTokens distribute accurate token quantities.

Status: OPEN

5- Title: sendLeftoverTokens Vulnerable to tokenInstance Swap

Severity: Medium

File and Code Location: ICO.sol(sendLeftoverTokens(), Lines 245, 247



```
function sendLeftoverTokens() external onlyOwner {
    require(
        isIcoEnded && isClaimEnabled,
        "ICO is not ended or claiming is not enabled."
    );
    require(
        tokenInstance != IERC20Metadata(address(0)),
        "Token address is not set."
    );
    uint256 amount = tokenInstance.balanceOf(address(this));
    require(amount > 0, "No tokens left to send.");
    TransferHelper.safeTransfer(address(tokenInstance), receiverAddress, amount);
}
```

Description: The sendLeftoverTokens function queries and transfers tokens based on the current tokenInstance. Since tokenInstance is arbitrarily mutable by the owner, a malicious swap could cause this function to sweep the wrong (e.g., worthless) token, effectively trapping or diverting legitimate leftover tokens from the project's treasury.

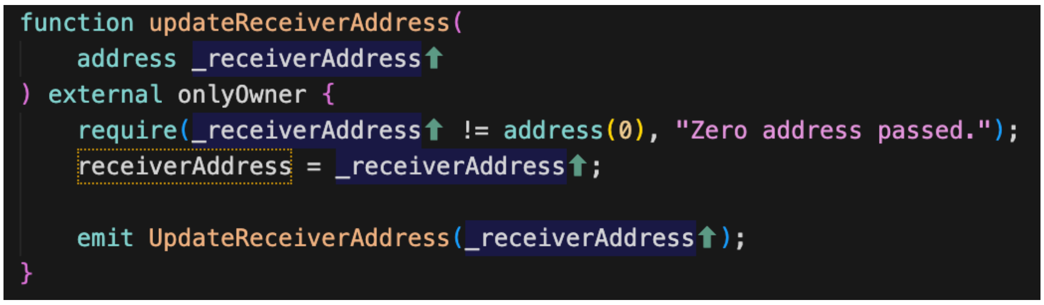
Mitigation: Make tokenInstance immutable after initialization, or implement timelocks/multi-sig for setTokenAddress changes. This prevents malicious swaps affecting leftover token sweeps.

Status: OPEN

6- Title: Unconstrained & Immediate Fund Redirection

Severity: Medium

File and Code Location: ICO.sol(updateReceiverAddress(), Line 256



```
function updateReceiverAddress(
    address _receiverAddress
) external onlyOwner {
    require(_receiverAddress != address(0), "Zero address passed.");
    receiverAddress = _receiverAddress;

    emit UpdateReceiverAddress(_receiverAddress);
}
```

Description: The updateReceiverAddress function allows the owner to instantly redirect all incoming ICO funds (ETH, ERC20) and leftover tokens to any address, without prior approval or delay. This grants unilateral power for arbitrary fund redirection.

Mitigation: Implement a timelock for updateReceiverAddress changes, or require multi-signature approval. This prevents immediate, unauthorized redirection and theft of all project funds.

Status: OPEN

7- Title: naccessible Tokens if ICO Token Address Unset

Severity: LOW

File and Code Location: ICO.sol - Line 122 (buyTokens allocation), and Lines 213-214 (claimTokens check), 197-200 (setTokenAddress).

Description: Users can successfully purchase tokens, but buyTokens lacks a check that tokenInstance (the actual ICO token contract address) is set. If unset, claimTokens will revert, making purchased tokens inaccessible until the owner manually configures tokenInstance.

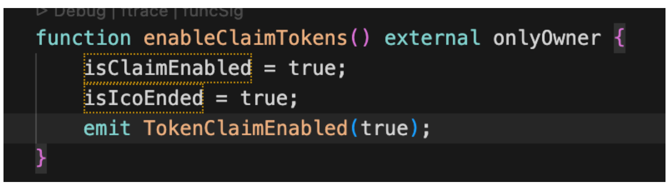
Mitigation: Add require(tokenInstance != address(0)) at the beginning of buyTokens. This prevents purchases until the ICO token address is properly configured by the owner.

Status: OPEN

8- Title: Prematurely Halts ICO Sales

Severity: LOW

File and Code Location: ICO.sol(enableClaimTokens), Line 193



```
function enableClaimTokens() external onlyOwner {
    isClaimEnabled = true;
    isIcoEnded = true;
    emit TokenClaimEnabled(true);
}
```

Description: The owner calling enableClaimTokens forcibly sets isIcoEnded=true, immediately stopping token purchases via buyTokens. This manual override creates an unclear ICO lifecycle, as sales can end prematurely even if the token cap isn't met, potentially confusing users and disrupting the intended sale duration.

Mitigation: Remove isIcoEnded = true; from enableClaimTokens to allow the token cap to solely control the sale end. Alternatively, add safeguards (e.g., a timelock) and clearly document this manual termination power.

Status: OPEN

DISCLAIMER

This audit report is not the final deployed version of the contract. So, any changes made after the audit will be considered out of the scope of the audit, and we will not be responsible for any changes related to that.