CHAINPROOF

Audit Report

AAMTOKEN

May 2025

Network BSC

Address 0xed7e3c10fba2fd235eac584c0868b0b45e40e4a2

Audited by © ChainProof

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	ROF	Redundant Ownership Functionality	Unresolved
●	MEM	Misleading Error Messages	Unresolved

●	MRF	Missing Renounce Functionality	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

Table of Contents

Risk Classification

The criticality of findings in ChainProof's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.

3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	AAMTOKEN
Compiler Version	v0.8.28+commit.7893614a
Optimization	200 runs
Explorer	https://bscscan.com/address/0xed7e3c10fba2fd235eac584c0868b0b45e40e4a2
Address	0xed7e3c10fba2fd235eac584c0868b0b45e40e4a2

Network	BSC
Symbol	AMTN
Decimals	18
Total Supply	3.000.000.000

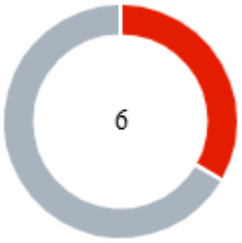
Audit Updates

Initial Audit	14 Apr 2025 https://github.com/ChainProof-io/audits/blob/main/amtn/v1/audit.pdf
Corrected Phase 2	28 Apr 2025 https://github.com/ChainProof-io/audits/blob/main/amtn/v2/audit.pdf
Corrected Phase 3	05 May 2025

Source Files

Filename	SHA256
contracts/AAMToken.sol	f2b48466c3482cdb16055fa513164ddce937477aaaa754b12ea4307e85d45014

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	4

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	4	0	0	0

MT - Mints Tokens

Criticality	Critical
Location	contracts/AAMToken.sol#L156

Status	Unresolved

Description

Users are able to mint amounts of tokens that are not tracked by the total supply. As a result, the contract tokens will be highly inflated.

In the `batchTransfer` users are able to add a list of `amounts` they want to transfer and recipients to receive them. The amounts are added to the balances of the recipients and are summed to calculate the `totalAmount` that will be subtracted by the balance of the sender. Since the entire process happens inside an `unchecked`, if the totalAmount surpasses the max unsigned integer, it will be wrapped to a lower value. This will result in the recipients receiving any amount of tokens as long as the wrapped `totalAmount` is not greater than the balance of the sender essentially minting an unlimited amount of tokens. This is also described in the `TSD` finding.

```
unchecked {     for (uint256 i = 0; i < len; ++i) {
address recipient = recipients[i];          uint256
amount = amounts[i];          require(recipient !=
address(0), "Zero address recipient");
require(recipient != sender, "Cannot transfer to
self");          totalAmount += amount;
_balances[recipient] += amount;          emit
Transfer(sender, recipient, amount);
    }
    require(totalAmount <= senderBalance,
"Insufficient balance");
    _balances[sender] = senderBalance - totalAmount;
}
```

Recommendation

The team should modify the `batchTransfer` function to avoid the minting of tokens and discrepancy between the sum of all balances and the total supply.

TSD - Total Supply Diversion

Criticality	Critical
Location	contracts/AAMToken.sol#L145,156,164,170
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specifically, in the `batchTransfer` method, users are able to add as parameter an array of amounts they want to transfer. If the `totalAmount` which is the combined value of the `amounts` array exceeds the max unsigned integer value, it will be wrapped to a lower value. This will result in users bypassing the requirement of `totalAmount <= senderBalance` as long as the final wrapped value is less than or equal to the `senderBalance`. This will allow users to increase account balances with amounts of tokens that are not reflected in the `totalSupply`. Since this function can be called by anyone, an unlimited amount of tokens can be generated resulting in possible price manipulation and loss of value.

```
unchecked {     for (uint256 i = 0; i <
len; ++i) {         address recipient =
recipients[i];         uint256 amount =
amounts[i];
         require(recipient != address(0), "Zero
address recipient");         require(recipient !=
sender, "Cannot transfer to self");
totalAmount += amount;

        _balances[recipient] += amount;
emit Transfer(sender, recipient, amount);
    }       require(totalAmount <= senderBalance,
"Insufficient balance");
    _balances[sender] = senderBalance - totalAmount;
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

ROF - Redundant Ownership Functionality

Criticality	Minor / Informative
Location	contracts/AAMToken.sol#L35,43,218
Status	Unresolved

Description

The contract implements functionality to define an owner. This functionality is normally implemented in contracts that need some form of authority and access control. However, excluding the `transferOwnership` there is no function in the contract that needs to be called only by the owner. Therefore the ownable functionality is redundant.

```
address public owner; modifier onlyOwner() {
require(msg.sender == owner, "Only owner");
    _; } function transferOwnership(address newOwner)
external onlyOwner
{        require(newOwner != address(0), "ERC20: transfer
ownership to the zero address");
        address oldOwner =
owner;      owner = newOwner;
        emit OwnershipTransferred(oldOwner,
newOwner);
}
```

Recommendation

It is recommended to remove the ownable functionality to increase code optimization and readability.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/AAMToken.sol#L219
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the actual implementation, making it difficult to understand the source code.

Specifically, in `transferOwnership` the requirement has an error message that mentions `ERC20:...` . `transferOwnership` is not a standard `ERC20` function. Defining error messages with `ERC20` at the start of the message is considered a best practice for token contracts in order to notify external parties that the error comes from an `ERC20` function.

```
require(newOwner != address(0), "ERC20: transfer ownership to
the zero address");
```

Recommendation

The team is advised to carefully review the error messages in order to reflect the actual implementation.

MRF - Missing Renounce Functionality

Criticality	Minor / Informative
Location	contracts/AAMToken.sol
Status	Unresolved

Description

When initialized the contract sets the `msg.sender` as the contracts `owner` . However the contract does not implement any functionality to renounce the ownership of the contract.

```
address public owner;
 modifier onlyOwner() {     require(msg.sender ==
owner, "Only owner");
    _;
}
```

Recommendation

While it is technically possible to transfer ownership to a contract wallet address that has zero functionality, simulating renouncing the ownership, it is recommended to add the necessary functionality for transferring it to address zero.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/AAMToken.sol#L31,32,33
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
string private constant _name = "AAMTOKEN"
string private constant _symbol = "AMTN"
uint8 private constant _decimals = 18
```

Recommendation

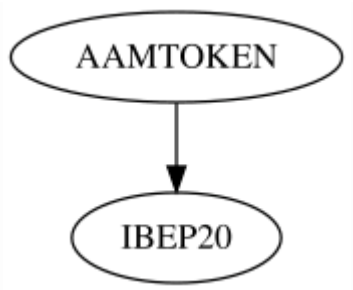
By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with. Find more information on the Solidity documentation https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

Functions Analysis

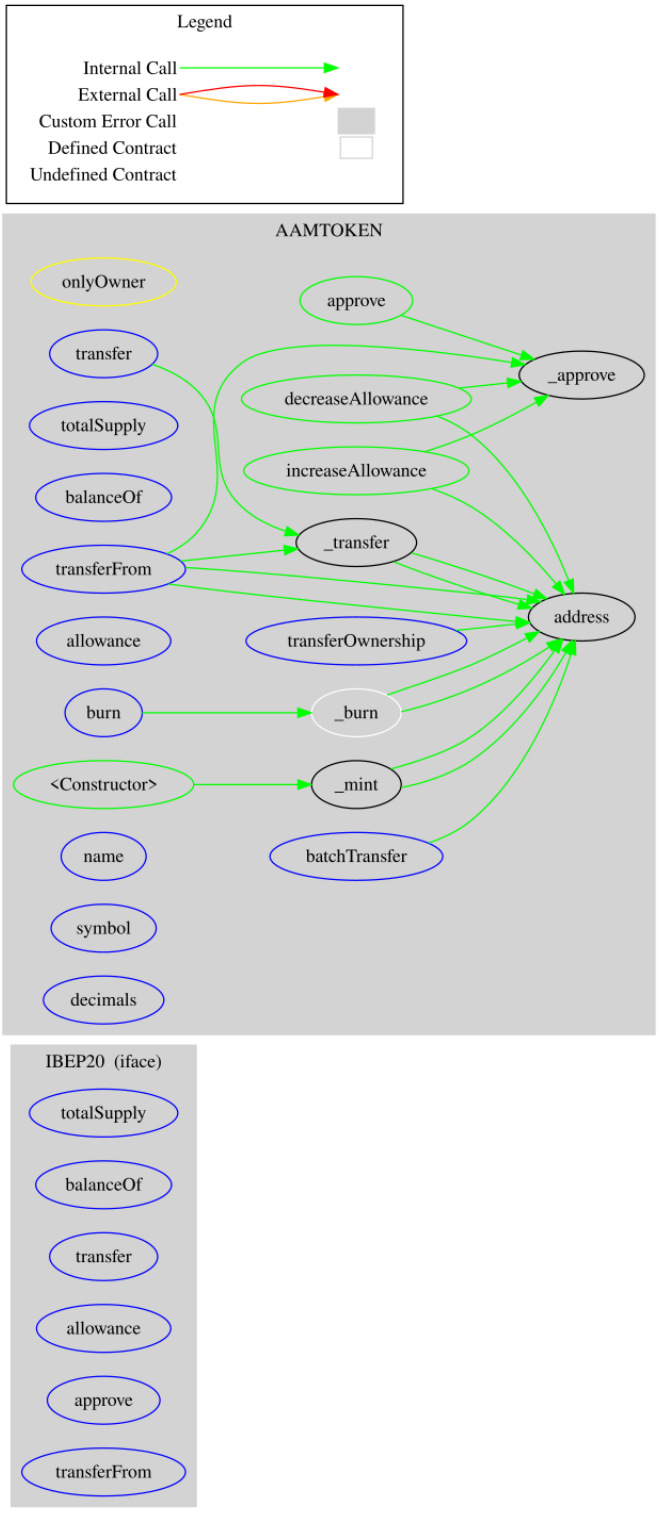
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AAMTOKEN	Implementation	IBEP20		
		Public	✓	-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	Public	✓	-
	_approve	Internal	✓	
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	transferFrom	External	✓	-
	batchTransfer	External	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	

	_burn	Internal	✓	
	burn	External	✓	-
	transferOwnership	External	✓	onlyOwner
	name	External		-
	symbol	External		-
	decimals	External		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
AAMTOKEN
onlyOwner
approve
transfer
_approve
decreaseAllowance
totalSupply
increaseAllowance
balanceOf
_transfer
transferFrom
address
allowance
transferOwnership
burn
_burn
<Constructor>
_mint
name
batchTransfer
symbol
decimals
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

AAMToken contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without ChainProof’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts ChainProof to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk ChainProof’s position is that each company and individual are responsible for their own due diligence and continuous security ChainProof’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by ChainProof are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About ChainProof

Chainproof is an Audit & KYC firm for Blockchain Projects, aimed at securing the Blockchain and the assets at risk. Chainproof is fueled by Industry grade experienced Blockchain Developers from all around the globe. From finding vulnerabilities, potential scams, malicious code mitigation, improper implementation of the token which can lead to loss of user's fund, you name it and we cover and secure them all.

Security testing and risk mitigation is given the highest priority at ChainProof. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.
ChainProof is aiming to make crypto discoverable and efficient globally. We associate with extremely robust testing and code review, leaving no room for any security risks because, when it comes to user's funds, we need to leave no stone unturned. Cheers!

The ChainProof team

ChainProof.dev