Cyberscope

Audit Report

XMAMA

November 2024

Network	BSC
Address	0xA7ad18090d35d3f00CfBBF927AceD534d01d74b1
Audited by	© cyberscope

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	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MTL	Missing Transfer Logic	Unresolved
●	UF	Unused Functions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope’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	XMAMA
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa7ad18090d35d3f00cfbbf927aced534d01d74b1
Address	0xa7ad18090d35d3f00cfbbf927aced534d01d74b1
Network	BSC
Symbol	xmam
Decimals	18
Total Supply	50,000,000,000
Badge Eligibility	Must Fix Criticals

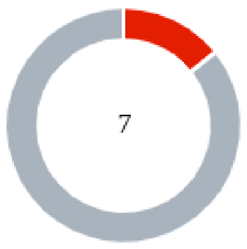
Audit Updates

Initial Audit	08 Nov 2024

Source Files

Filename	SHA256
XMAMA.sol	5c68b4191520812091b307e9a30348d26c99caccf6abd24b8ba6423ca13a6b15

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	6

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	6	0	0	0

MTL - Missing Transfer Logic

Criticality	Critical
Location	XMAMA.sol#L1077
Status	Unresolved

Description

The contract contains a `transfer` function, which is designed to facilitate the transfer of tokens from one address to another. However, this function lacks the core logic required to execute a token transfer. As a result, any user attempting to use this function will find that no tokens are actually moved to the intended recipient address, despite the function being present in the contract. This omission not only renders the function ineffective but also creates a misleading interface that could lead to confusion or loss of trust from users who expect a functioning transfer mechanism.

```
function transfer(address to, uint256 amount)
    public
    virtual
    override
    whenNotPaused
    validateTransfer(msg.sender, to)
    returns (bool)
  {

  }
```

Recommendation

It is recommended to implement the necessary logic within the `transfer` function to correctly handle the transfer of tokens. This includes deducting the specified `amount` from the sender’s balance and crediting it to the recipient’s address, while ensuring proper event emissions and balance checks.

UF - Unused Functions

Criticality	Minor / Informative
Location	XMAMA.sol#L862,988
Status	Unresolved

Description

The contract includes an `ERC20ConfigProps` struct designed to manage key configurations for the token's functionalities, such as enabling token burning, pausing, blacklisting, whitelisting, and force transfers. However, each property within this struct is initialized to `false` by default and remains in this state throughout the contract's lifecycle. Notably, there is no setter function to modify these properties, resulting in a static configuration where all functionalities are effectively disabled. Consequently, functions that rely on these properties, such as `isBurnable`, `isBlacklistEnabled`, `isWhitelistEnabled`, and `isForceTransferAllowed`, are redundant, as they merely return the unchangeable `false` values. This configuration could mislead users into believing these functionalities exist when, in reality, they are permanently inactive.

```
 struct ERC20ConfigProps {
    bool _isBurnable;
    bool _isPausable;
    bool _isBlacklistEnabled;
    bool _isWhitelistEnabled;
    bool _isForceTransferAllowed;
  }

  /// @notice method which checks if the token is burnable
  function isBurnable() public view returns (bool) {
    return configProps._isBurnable;
  }

  /// @notice method which checks if the token supports blacklisting
  function isBlacklistEnabled() public view returns (bool) {
    return configProps._isBlacklistEnabled;
  }

  /// @notice method which checks if the token supports whitelisting
  function isWhitelistEnabled() public view returns (bool) {
    return configProps._isWhitelistEnabled;
  }

  /// @notice method which checks if the token supports force transfers
  function isForceTransferAllowed() public view returns (bool) {
    return configProps._isForceTransferAllowed;
  }
```

Recommendation

It is recommended to remove the unused configuration properties within the `ERC20ConfigProps` struct, and the corresponding functions, as they do not contribute to the contract's functionality. Alternatively, if these features are intended for future use, ensure that the values are appropriately initialized and utilized within the contract's logic to accurately represent the supported capabilities.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	XMAMA.sol#L33
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function safeTransferETH(address to, uint256 amount) internal {
    bool success;
    // solhint-disable-next-line no-inline-assembly
    assembly {
      // Transfer the ETH and store if it succeeded or not.
      success := call(gas(), to, amount, 0, 0, 0, 0)
    }
    if (!success) {
      revert PaymentFailed(to, amount);
    }
  }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	XMAMA.sol#L953
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
initialTokenOwner = tokenOwner
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	XMAMA.sol#L36
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
      // Transfer the ETH and store if it succeeded or not.
      success := call(gas(), to, amount, 0, 0, 0, 0)
    }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	XMAMA.sol#L9,51,78,185,270,355,385,770,805,845
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity 0.8.7;
pragma solidity ^0.8.0;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	XMAMA.sol#L51,78,185,270,355,385,770,805
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

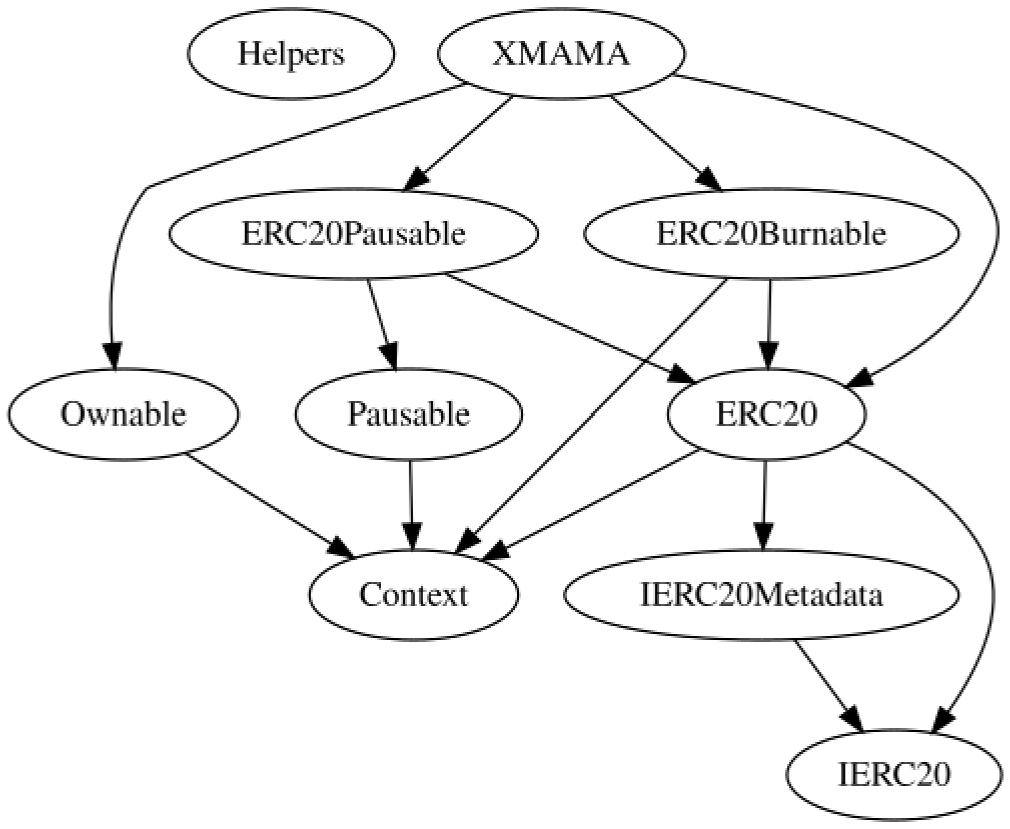
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

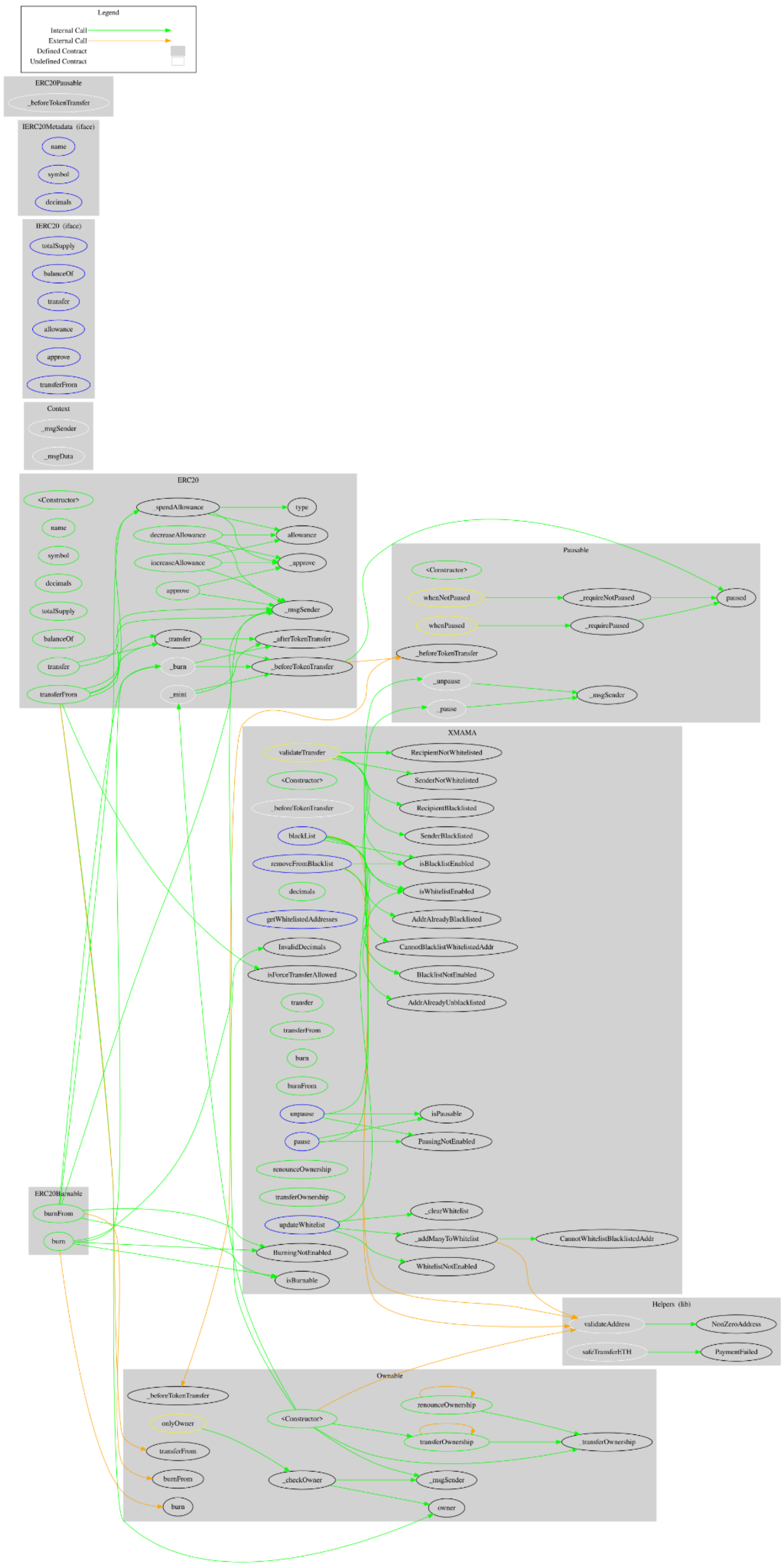
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
XMAMA	Implementation	ERC20, ERC20Burnable, ERC20Pausable, Ownable		
		Public	✓	ERC20
	_beforeTokenTransfer	Internal	✓	
	isPausable	Public		-
	isBurnable	Public		-
	isBlacklistEnabled	Public		-
	isWhitelistEnabled	Public		-
	isForceTransferAllowed	Public		-
	decimals	Public		-
	getWhitelistedAddresses	External		-
	blackList	External	✓	onlyOwner whenNotPaused
	removeFromBlacklist	External	✓	onlyOwner whenNotPaused
	transfer	Public	✓	whenNotPaused validateTransfer
	transferFrom	Public	✓	whenNotPaused validateTransfer
	burn	Public	✓	onlyOwner whenNotPaused

	burnFrom	Public	✓	onlyOwner whenNotPaused
	pause	External	✓	onlyOwner
	unpause	External	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner whenNotPaused
	transferOwnership	Public	✓	onlyOwner whenNotPaused
	updateWhitelist	External	✓	onlyOwner
	_addManyToWhitelist	Private	✓	
	_clearWhitelist	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20Pausable
_beforeTokenTransfer
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
transferFrom
spendAllowance
decreaseAllowance
increaseAllowance
approve
_transfer
_burn
_mint
type
allowance
_approve
_msgSender
_afterTokenTransfer
_beforeTokenTransfer
Pausable
<Constructor>
whenNotPaused
whenPaused
_beforeTokenTransfer
_unpause
_pause
_requireNotPaused
_requirePaused
_msgSender
paused
XMAMA
validateTransfer
<Constructor>
_beforeTokenTransfer
blackList
removeFromBlacklist
decimals
getWhitelistedAddresses
InvalidDecimals
isForceTransferAllowed
transfer
transferFrom
burn
burnFrom
unpause
pause
renounceOwnership
transferOwnership
updateWhitelist
BurningNotEnabled
isBurnable
RecipientNotWhitelisted
SenderNotWhitelisted
RecipientBlacklisted
SenderBlacklisted
isBlacklistEnabled
isWhitelistEnabled
AddrAlreadyBlacklisted
CannotBlacklistWhitelistedAddr
BlacklistNotEnabled
AddrAlreadyUnblacklisted
isPausable
PausingNotEnabled
_clearWhitelist
_addManyToWhitelist
WhitelistNotEnabled
CannotWhitelistBlacklistedAddr
ERC20Burnable
burnFrom
burn
Helpers (lib)
validateAddress
safeTransferETH
NonZeroAddress
PaymentFailed
Ownable
_beforeTokenTransfer
onlyOwner
transferFrom
burnFrom
burn
<Constructor>
_checkOwner
renounceOwnership
transferOwnership
_transferOwnership
_msgSender
owner

Summary

XMAMA contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. XMAMA is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error but 1 critical issue. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users’ transactions.

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 Cyberscope'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 Cyberscope 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope 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 Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io