Cyberscope

Audit Report

Xenum Space

February 2023

Type	BEP20
Network	BSC
Address	0x0f5f74e6e23A3fDa44B33Cd4C658EfA070b3cAFC
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Xenum
Compiler Version	v0.8.10+commit.fc410830
Optimization	200 runs
Explorer	https://bscscan.com/address/0x0f5f74e6e23a3fda44b33cd4c658efa070b3cafc
Address	0x0f5f74e6e23a3fda44b33cd4c658efa070b3cafc
Network	BSC
Symbol	XNM
Decimals	18
Total Supply	999,364,041

Audit Updates

Initial Audit	01 Mar 2023

Source Files

Filename	SHA256
Xenum.sol	5c47469bf0c6db089015afc1102c03637f94268f45b519373e3eb45748343408

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Xenum.sol#L91,95
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `SetMarketingPercent` and `SetBurnPercent` functions with a high percentage value.

```
function SetMarketingPercent(uint256 _marketingPercent) onlyOwner public {
    marketingPercent = _marketingPercent;
}

function SetBurnPercent(uint256 _burnPercent) onlyOwner public {
    burnPercent = _burnPercent;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	PRE	Potential Reentrance Exploit	Unresolved
Critical	TSD	Total Supply Diversion	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L06	Missing Events Access Control	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L11	Unnecessary Boolean equality	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

PRE - Potential Reentrance Exploit

Criticality	Critical
Location	Xenum.sol#L297
Status	Unresolved

Description

As part of the `fallback` method, the contract transfers ETH to the user if the amount of tokens is greater than the balance of the `CrowdAdress`. Since the user can be any address, the address can be exploited for a re-entrance attack. The `CrowdAdress` will still have the entire balance during the re-entrance phase since the amount has not yet been subtracted.

```
fallback() external payable
{
    require(CrowdSupply > 0);
    require(PresaleStart == true, "PreSale is not active.");
    uint256 tokensPerOneEther = Price;
    uint256 tokens = tokensPerOneEther * msg.value / 10**18;
    if (tokens > _balances[CrowdAdress])
    {
        tokens = _balances[CrowdAdress];
        uint valueWei = tokens * 10**18 / tokensPerOneEther;
        payable(msg.sender).transfer(msg.value - valueWei);
    }
    require(tokens > 0);

    _balances[msg.sender] += tokens;
    _balances[CrowdAdress] -= tokens;
    CrowdSupply -= tokens;
    PresaleBalance += msg.value;
    emit Transfer(CrowdAdress, msg.sender, tokens);
}
```

Recommendation

The team is advised to prevent the re-entrance exploit as part of the solidity best practices. Some suggestions are:

- Not allow contract addresses to receive funds.
- Add a locker/mutex in the transfer method scope.
- Transfer the funds as the last statement of the transfer method, so that the balance will have been subtracted during the re-entrance phase.

TSD - Total Supply Diversion

Criticality	Critical
Location	Xenum.sol#L222
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, during transactions, a portion of the transferred amount is added to the balance of the `burnAddress` and then subtracted from the `_totalSupply`. As a result, the sum of balances is diverse from the total supply.

```
if (burnPercent > 0)
{
    _balances[burnAddress] += burnAmount;
    _totalSupply -= burnAmount;
}
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Xenum.sol#L74,76,265,266,276
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
string public _name = "Xenum"
string public _symbol = "XNM"
uint256 public aAmt          =    1000*10**18
uint256 public Price         =    300000*10**18
bool public PresaleBayback = false
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Xenum.sol#L68,70,72,74,76,87,91,95,109,262,263,264,266,269,271,274,275,276,278,283
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.

```
mapping(address => uint256) public _balances
mapping(address => mapping(address => uint256)) public _allowances
uint256 public _totalSupply = 1000000000*10**18
string public _name = "Xenum"
string public _symbol = "XNM"

function SetMarketingAddress(address payable  _marketingAddress) onlyOwner public {
        marketingAddress = _marketingAddress;
    }
address payable  _marketingAddress
uint256 _marketingPercent

function SetMarketingPercent(uint256 _marketingPercent) onlyOwner public {
        marketingPercent = _marketingPercent;
    }

...
```

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/v0.8.17/style-guide.html#naming-convention.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	Xenum.sol#L111
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
owner = _owner
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Xenum.sol#L229
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 _burn(address account, uint256 amount) internal virtual
    {
        require(account != address(0), "ERC20: burn from the zero address");
        _beforeTokenTransfer(account, address(0), amount);
        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked
        {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;
        emit Transfer(account, address(0), 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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Xenum.sol#L300,321,322
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(PresaleStart == true, "PreSale is not active.")
require(AirdropStart == true, "Airdrop is not active.")
require(_AirdropsDone[msg.sender] == false, "Airdrop already received")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Xenum.sol#L302,306
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 tokens = tokensPerOneEther * msg.value / 10**18
uint valueWei = tokens * 10**18 / tokensPerOneEther
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Xenum.sol#L88,111
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.

```
marketingAddress = _marketingAddress
owner = _owner
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Xenum.sol#L1
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 ^0.8.10;
```

Recommendation

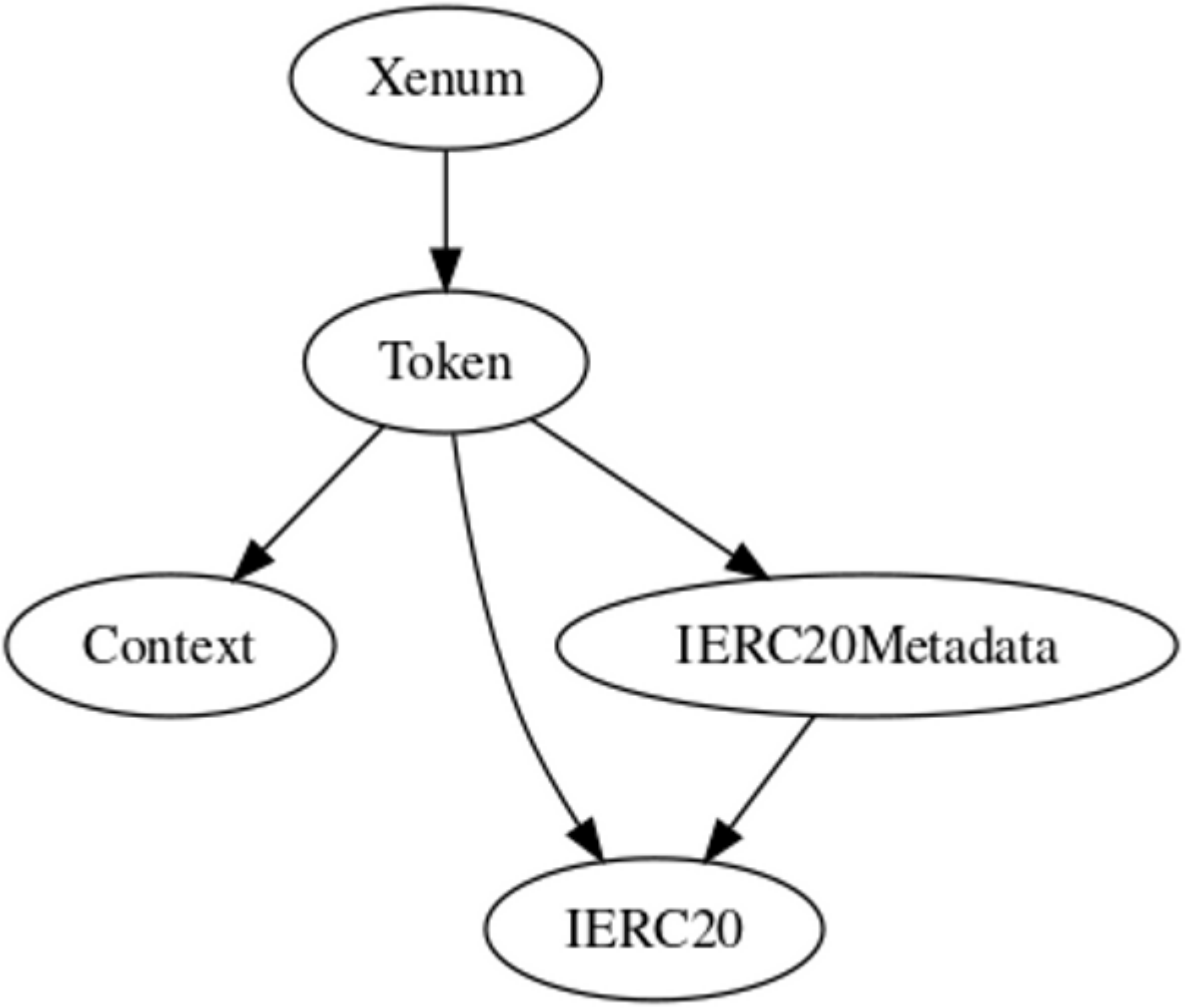
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Token	Implementation	Context, IERC20, IERC20Metadata		
	SetMarketingAddress	Public	✓	onlyOwner
	SetMarketingPercent	Public	✓	onlyOwner
	SetBurnPercent	Public	✓	onlyOwner
	changeOwner	Public	✓	onlyOwner

	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
Xenum	Implementation	Token		
	PresaleOnOf	Public	✓	onlyOwner
	AirdropOnOf	Public	✓	onlyOwner
		Public	✓	-
		External	Payable	-
	getAirdrop	Public	✓	-
	withdraw	Public	Payable	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Xenum
PresaleOnOf
AirdropOnOf
<Constructor>
<Fallback>
getAirdrop
withdraw
payable
Token
SetMarketingAddress
SetMarketingPercent
SetBurnPercent
onlyOwner
changeOwner
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_transfer
_msgSender
_approve
_burn
_beforeTokenTransfer
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

Summary

Xenum Space contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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

https://www.cyberscope.io