Cyberscope

Audit Report

X

July 2023

Network BSC

Address 0xb0C5E1fAFe22bDD9e44464eF5EB9451108318D4e

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	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CR	Code Repetition	Unresolved
●	RCS	Redundant Code Statements	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Explorer	https://bscscan.com/address/0xb0c5e1fafe22bdd9e44464ef5eb9451108318d4e

Audit Updates

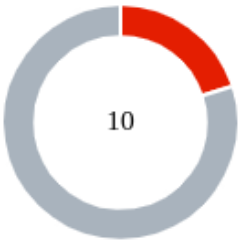
Initial Audit	31 Jul 2023

Source Files

Filename	SHA256
contracts/XV4.sol	3d23e5f274fe46103ae6d128aa19b4baa8e3767501d9e207cc952f292b551730
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	db92fc1b515decad3a783b1422190877d2d70b907c6e36fb0998d9465aee42db
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	78a6bc84bbb417f0d8a6b12e181e0f783151774f4f0c054c5d3f920e70d69f8c
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	9619cf23b549a5126042a4e20b09a2eb12dc8c2975258e3b8cde79cc593b6926
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol	68bcca423fc72ec9625e219c9e36306c726a347e43f3711467c579bd3f6500c8
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol	ca660e828b0c4be205a9f56f3b87b91c1fa67cfd0f6e9dbd431faea7a6280d36

@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a2c4e5c274a586f145d278293ae33198cd8f412ab7e6d26f2394c8949b32b24b
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	1fbf2a131b895514f0027866cc0deff151ea16424b4aed2b8c573d2275cfa9e8

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

MT - Mints Tokens

Criticality	Critical
Location	contracts/XV4.sol#L40
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

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

Recommendation

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.

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/XV4.sol#L51,90
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `addBlackList` function.

```
function transfer(address _to, uint _value) public override
returns (bool) {
    require(!isBlackListed[msg.sender]);
    ...
}

function addBlackList(address _evilUser) public onlyOwner {
    isBlackListed[_evilUser] = true;
    emit AddedBlackList(_evilUser);
}
```

Recommendation

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/XV4.sol#L30,44
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specifically, both the `initialize` and `setup` function, sets the same variables: `_TIMELOCK` , `_ALLOW_AMOUNT` , and `PancakeSwapRouter` to identical values. The `initialize` function is presumably used to set initial values for these variables when the contract is first initialized. On the other hand, the `setup` function, performs the same operations, setting the aforementioned variables to the same values as the `initialize` function. This repetition is unnecessary and can lead to confusion, as it's unclear why there would be a need to reset these variables to their initial values after the contract has already been initialized.

```
    function initialize() public initializer {
        ...
        _TIMELOCK = 1 days;
        _ALLOW_AMOUNT = 500;
        PancakeSwapRouter =
0x10ED43C718714eb63d5aA57B78B54704E256024E;
        ...
    }

    function setup() public onlyOwner {
        _TIMELOCK = 1 days;
        _ALLOW_AMOUNT = 500;
        PancakeSwapRouter =
0x10ED43C718714eb63d5aA57B78B54704E256024E;
    }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, it is recommended to remove the `setup` function if its sole purpose is to reset the variables to their initial values, as this operation is already performed by the `initialize` function. If the `setup` function has other purposes or functionalities not shown in the provided code snippet, it would be advisable to refactor it to avoid redundancy. By doing so, the contract will be more concise, easier to maintain, and less prone to potential errors or vulnerabilities stemming from repetitive code.

RCS - Redundant Code Statements

Criticality	Minor / Informative
Location	contracts/XV4.sol#L20,100,105,110,115,119,123
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract sets the variables `_TIMELOCK` and `_ALLOW_AMOUNT`, and using several functions, namely `addWhiteList`, `removeWhiteList`, `setNewPersentage`, `setNewTime`, and `timeAndAmount`. These functions are designed to set, modify, and retrieve certain variables and states within the contract. However, these functions and variables are not utilized in the actual execution or logic of the contract. The only instances where these functions are referenced are within commented-out sections of the code. This means that the actual implementation of the contract does not take these functions into consideration, rendering them redundant.

```
    uint _TIMELOCK;
     uint _ALLOW_AMOUNT;

    function addWhiteList(address _evilUser) public onlyOwner {
        isWhiteListed[_evilUser] = true;
        emit AddedWhiteList(_evilUser);
    }

    function removeWhiteList(address _clearedUser) public
onlyOwner {
        isWhiteListed[_clearedUser] = false;
        emit RemovedWhiteList(_clearedUser);
    }

    function setNewPersentage(uint _new) public onlyOwner {
        _ALLOW_AMOUNT = _new;
        _TIMELOCK = 1 days;
    }

    function setNewTime(uint _new) public onlyOwner {
        _TIMELOCK = _new;
    }

    function timeAndAmount() public view returns (uint, uint) {
        return (_TIMELOCK, _ALLOW_AMOUNT);
    }
```

Recommendation

The team is advised to remove the aforementioned code segments from the contract. Retaining unused code can lead to confusion, increased gas costs, and potential vulnerabilities in the future. By removing these redundant functions, the contract can be streamlined, making it more efficient and easier to understand.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/XV4.sol#L29
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.

```
address new_PancakeSwapRouter
```

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	contracts/XV4.sol#L20,21,22,29,50,82,83,84,90,95,100,105,110,115
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.

```
uint _TIMELOCK
uint _ALLOW_AMOUNT
address PancakeSwapRouter
address new_PancakeSwapRouter
uint _value
address _to
address _from
address _evilUser
address _clearedUser
uint _new
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	contracts/XV4.sol#L29
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
address new_PancakeSwapRouter
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/XV4.sol#L111,116
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.

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.

```
_ALLOW_AMOUNT = _new
_TIMELOCK = _new
```

Recommendation

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 with its arithmetic.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/XV4.sol#L125
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 {
            size := extcodesize(addr)
        }
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/XV4.sol#L2
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;
```

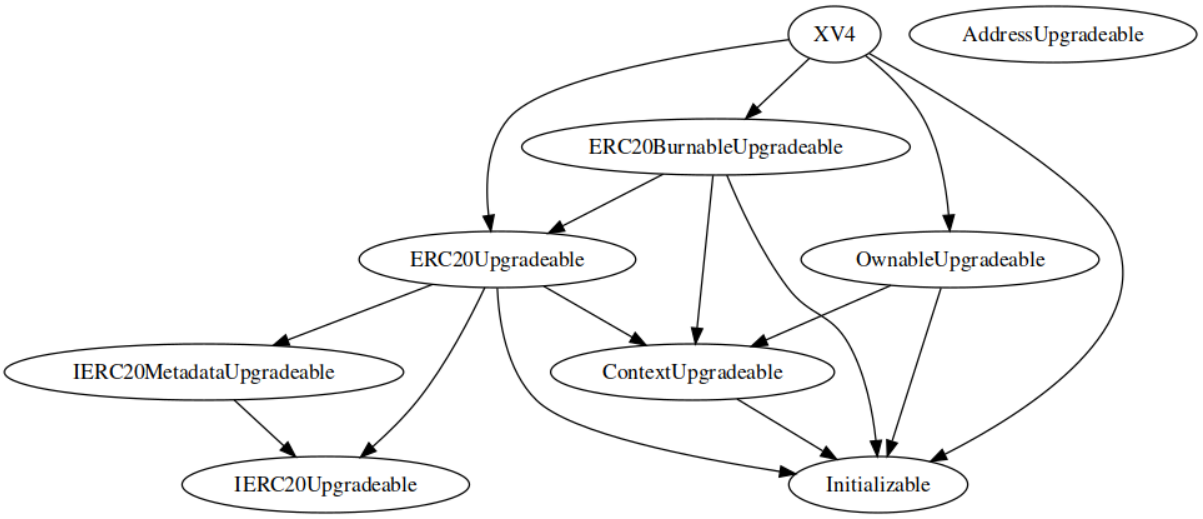
Recommendation

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.

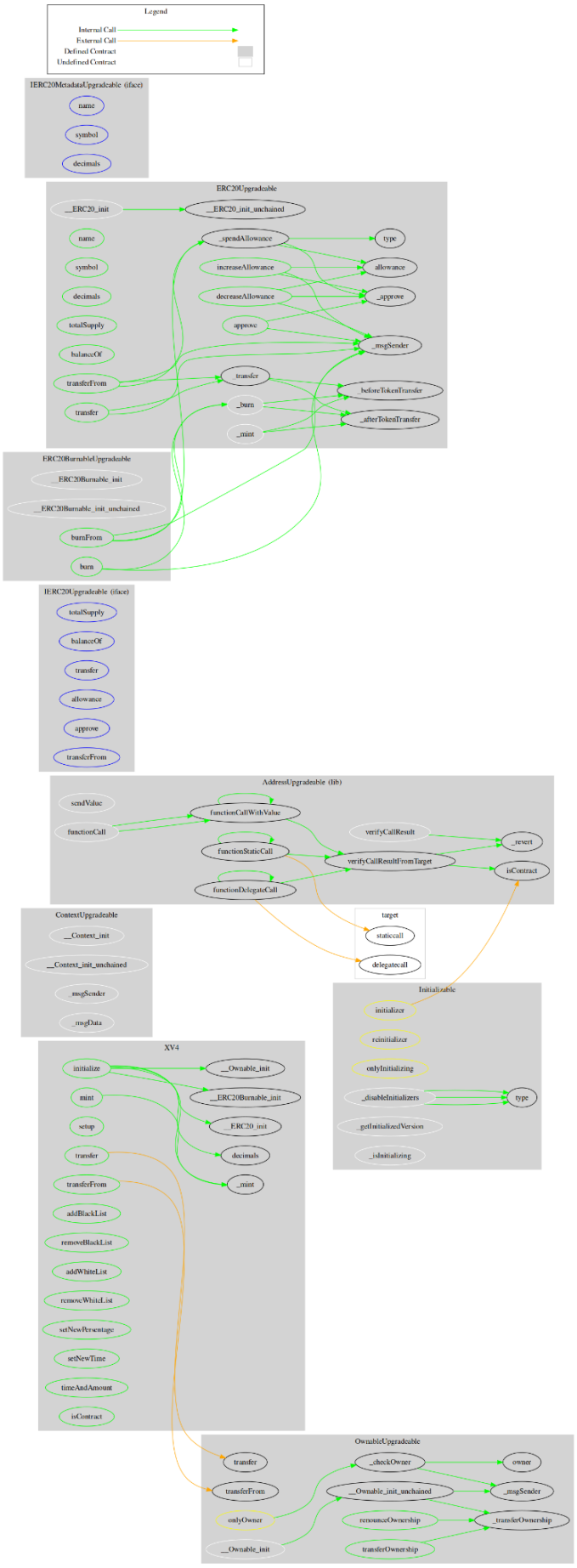
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
XV4	Implementation	Initializable, ERC20Upgradeable, ERC20BurnableUpgradeable, OwnableUpgradeable		
	initialize	Public	✓	initializer
	mint	Public	✓	onlyOwner
	setup	Public	✓	onlyOwner
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	addBlackList	Public	✓	onlyOwner
	removeBlackList	Public	✓	onlyOwner
	addWhiteList	Public	✓	onlyOwner
	removeWhiteList	Public	✓	onlyOwner
	setNewPersentage	Public	✓	onlyOwner
	setNewTime	Public	✓	onlyOwner
	timeAndAmount	Public		-
	isContract	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20MetadataUpgradeable (iface)
name
symbol
decimals
ERC20Upgradable
__ERC20_init
__ERC20_init_unchained
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
increaseAllowance
decreaseAllowance
approve
transfer
_burn
_mint
type
allowance
_approve
_msgSender
beforeTokenTransfer
_afterTokenTransfer
ERC20BurnableUpgradeable
__ERC20Burnable_init
__ERC20Burnable_init_unchained
burnFrom
burn
IERC20Upgradeable (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
AddressUpgradeable (lib)
sendValue
functionCall
functionCallWithValue
functionStaticCall
functionDelegateCall
verifyCallResult
verifyCallResultFromTarget
_revert
isContract
target
staticcall
delegatecall
ContextUpgradeable
__Context_init
__Context_init_unchained
_msgSender
_msgData
Initializable
initializer
reinitializer
onlyInitializing
_disableInitializers
_getInitializedVersion
_isInitializing
type
XV4
initialize
mint
setup
transfer
transferFrom
addBlackList
removeBlackList
addWhiteList
removeWhiteList
setNewPersentage
setNewTime
timeAndAmount
isContract
__Ownable_init
__ERC20Burnable_init
__ERC20_init
decimals
_mint
OwnableUpgradeable
transfer
transferFrom
onlyOwner
__Ownable_init
_checkOwner
__Ownable_init_unchained
renounceOwnership
transferOwnership
owner
_msgSender
_transferOwnership

Summary

X 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 mint tokens and massively blacklist addresses. If the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Initial Audit, 31 July 2023

At the time of the audit report, the contract with address

0xb0C5E1fAFe22bDD9e44464eF5EB9451108318D4e is pointed by the following proxy

address: 0x361dF081e8AcCac1115693b2E84a1A09d9F0d49c.

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