Cyberscope

Audit Report

Banana

January 2022

Type	BEP20
Network	BSC
Address	0x5F42F5b4E497195772cc23181F0EAA54cF0AA824
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Banana
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0x5f42f5b4e497195772cc23181f0eaa54cf0aa824
Address	0x5f42f5b4e497195772cc23181f0eaa54cf0aa824
Network	BSC
Symbol	BNS
Decimals	18

Audit Updates

Initial Audit	23 Nov 2022
Corrected Phase 2	19 Jan 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef8c4e410ae99608be5964d98fa701
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
contracts/ApesParadise/Banana.sol	9da6cbcc80ae45414a143e2a163f39fc24277786e85296eefef4b39fdaf9b4fc

Contract 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	Unresolved
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Passed

MT - Mints Tokens

Criticality	Critical
Location	contracts/apeparadise/Banana.sol#L49,55,61,69
Status	Unresolved

Description

The `forestAddress` address has the authority to mint tokens. The `forestAddress` address 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) external {
    require(forestAddress != address(0) && apeAddress != address(0) &&
caveAddress != address(0) && upgradeAddress != address(0), "missing initial
requirements");
    require(_msgSender() == forestAddress,"msgsender does not have
permission");
    _mint(_to, _amount);

}
```

The owner has the authority to mint tokens with three additional ways

```
function mintPromotionalBanana(address _to) external onlyOwner {}
function mintBnbLPBanana() external onlyOwner {}

function mintTreeLPBanana() external onlyOwner {}
```

Recommendation

The `forestAddress` address and owner should carefully manage the credentials. We advised considering an extra-strong security mechanism that the actions may be quarantined by many users instead of one. The owner could also renounce the contract ownership for a period of time or pass the access to the zero address.

BT - Burns Tokens

Criticality	Critical
Location	contracts/apeparadise/Banana.sol#L75
Status	Unresolved

Description

The `apeAddress`, `caveAddress,` and `upgradeAddress` have the authority to burn tokens from a specific address. They may take advantage of it by calling the burn function. As a result, the targeted contract address will lose the corresponding tokens.

```
function burn(address _from, uint256 _amount) external {
    require(apeAddress != address(0) && caveAddress != address(0) &&
upgradeAddress != address(0), "missing initial requirements");
    require(
        _msgSender() == apeAddress
        || _msgSender() == caveAddress
        || _msgSender() == upgradeAddress,
        "msgsender does not have permission"
    );
    _burn(_from, _amount);

}
```

Recommendation

The `apeAddress`, `caveAddress,` and `upgradeAddress` addresses should carefully manage the credentials of the owner's account. We advised considering an extra-strong security mechanism that the actions may be quarantined by many users instead of one. The owner could also renounce the contract ownership for a period of time or pass the access to the zero address.

OTUT - Transfers User's Tokens

Criticality	Critical
Location	contracts/apeparadise/Banana.sol#L86,92
Status	Unresolved

Description

The `caveAddress` address has the authority to transfer the balance of a user's contract to the `caveAddress` address. The `caveAddress` address may take advantage of it by calling the transferToCave function.

```
function transferToCave(address _from, uint256 _amount) external {
    require(caveAddress != address(0), "missing initial requirements");
    require(_msgSender() == caveAddress, "only the cave contract can call
transferToCave");
    _transfer(_from, caveAddress, _amount);

}
```

The upgradeAddress address has the authority to transfer the balance of a user's contract to the upgradeAddress address. The upgradeAddress address may take advantage of it by calling the transferForUpgradesFees function.

```
function transferForUpgradesFees(address _from, uint256 _amount) external {
    require(upgradeAddress != address(0), "missing initial requirements");
    require(_msgSender() == upgradeAddress, "only the upgrade contract can
call transferForUpgradesFees");
    _transfer(_from, upgradeAddress, _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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Contract Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	L18	Multiple Pragma Directives	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/ApesParadise/Banana.sol#L2
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 ^0.8.0;
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/ApesParadise/Banana.sol#L12
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.

```
uint256 public NUM_BANANA_BNB_LP = 50_000_000
```

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/ApesParadise/Banana.sol#L12,28,32,36,44,49,69,75,86,92
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.

```
uint256 public NUM_BANANA_BNB_LP = 50_000_000
address _forestAddress
address _caveAddress
address _upgradeAddress
address _apeAddress
address _to
uint256 _amount
address _from
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/ApesParadise/Banana.sol#L29,33,37,46
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.

```
forestAddress = _forestAddress
caveAddress = _caveAddress
upgradeAddress = _upgradeAddress
apeAddress = _apeAddress
```

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	contracts/ApesParadise/Banana.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 ^0.8.4;
```

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.

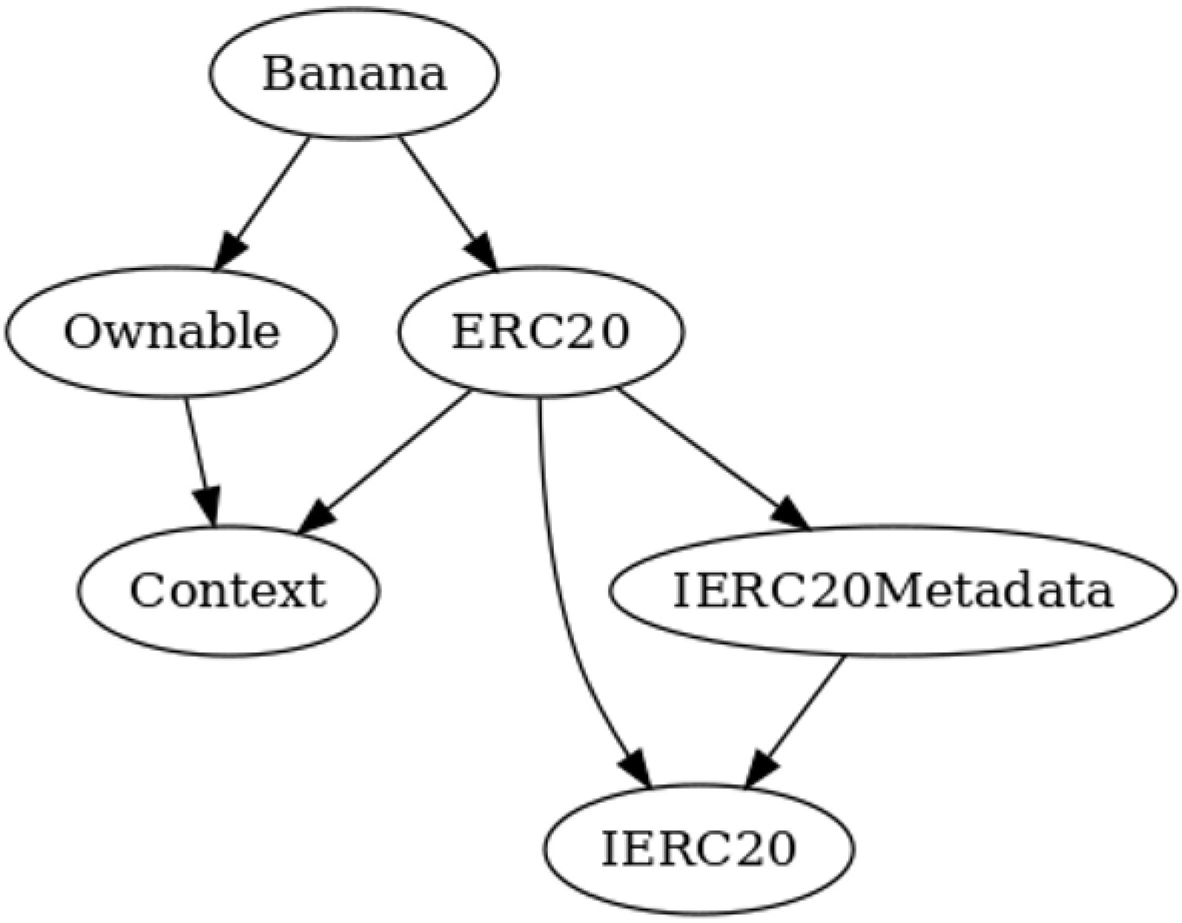
Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	

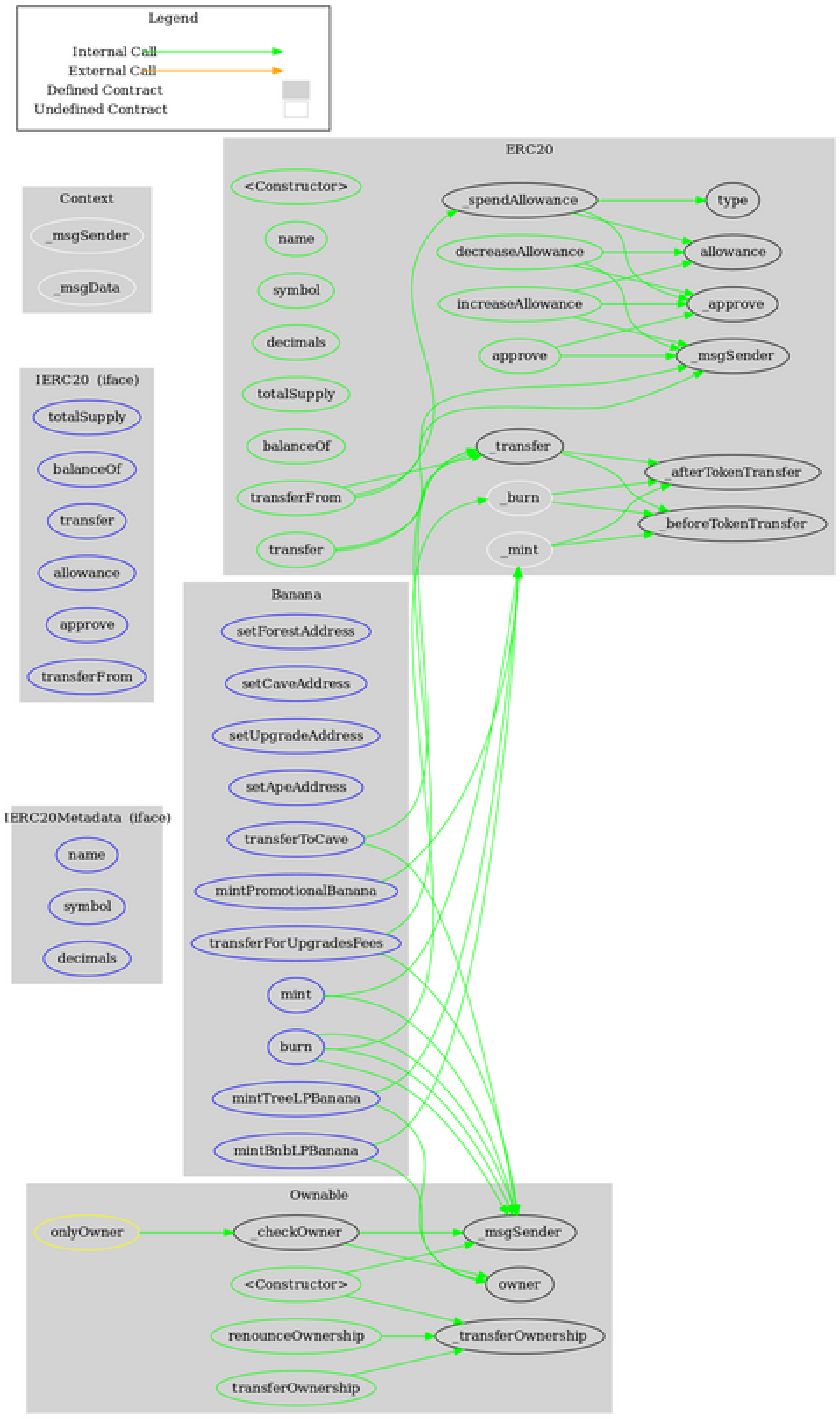
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Banana	Implementation	ERC20, Ownable		
	setForestAddress	External	✓	onlyOwner
	setCaveAddress	External	✓	onlyOwner
	setUpgradeAddress	External	✓	onlyOwner
	setApeAddress	External	✓	onlyOwner

	mintPromotionalBanana	External	✓	onlyOwner
	mintBnbLPBanana	External	✓	onlyOwner
	mintTreeLPBanana	External	✓	onlyOwner
	mint	External	✓	-
	burn	External	✓	-
	transferToCave	External	✓	-
	transferForUpgradesFees	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Context
_msgSender
_msgData
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
decreaseAllowance
increaseAllowance
approve
type
allowance
_approve
_msgSender
_transfer
_burn
_mint
_afterTokenTransfer
_beforeTokenTransfer
Banana
setForestAddress
setCaveAddress
setUpgradeAddress
setApeAddress
transferToCave
mintPromotionalBanana
transferForUpgradesFees
mint
burn
mintTreeLPBanana
mintBnbLPBanana
Ownable
onlyOwner
_checkOwner
<Constructor>
renounceOwnership
transferOwnership
_msgSender
owner
_transferOwnership

Summary

There are some functions that can be abused by the owner like transferring the user's tokens, minting tokens, and burning tokens. if the contract owner abuses the mint functionality, then the contract will be highly inflated. if the contract owner abuses the burning functionality, then the users could lose their tokens. 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

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