Cyberscope

Audit Report

Passionate kitten

November 2024

Network BSC

Address 0x8A916CF2184d749614E2DD7339b7176512F6F778

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	Unresolved
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CR	Code Repetition	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	RF	Redundant Functionalities	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L16	Validate Variable Setters	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	PassionateKitten
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x8a916cf2184d749614e2dd7339b7176512f6f778
Address	0x8a916cf2184d749614e2dd7339b7176512f6f778
Network	BSC
Symbol	PTM
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Must Fix Criticals

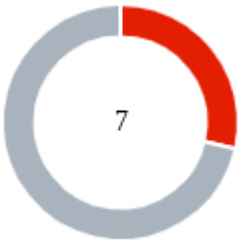
Audit Updates

Initial Audit	22 Nov 2024

Source Files

Filename	SHA256
PassionateKitten.sol	4d80fb2512fd59e966670ac8815e6509fb5d41e528d9721b0612b7abcef9242c
lib/LibCommon.sol	ad40e79524942f0927be19739e7c96b7a52147f5cf54fdd7eb676720db70b66a

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	5	0	0	0

MT - Mints Tokens

Criticality	Critical
Location	PassionateKitten.sol#L508
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) external onlyOwner
whenNotPaused {
    if (!isMintable()) {
      revert MintingNotEnabled();
    }
    if (isMaxAmountOfTokensSet()) {
      if (balanceOf(to) + amount > maxTokenAmountPerAddress) {
        revert DestBalanceExceedsMaxAllowed(to);
      }
    }
    if (isBlacklistEnabled()) {
      if (_isBlacklisted[to]) {
        revert RecipientBlacklisted(to);
      }
    }
    if (isWhitelistEnabled()) {
      if (!whitelist[to]) {
        revert RecipientNotWhitelisted(to);
      }
    }

    super._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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

BT - Burns Tokens

Criticality	Critical
Location	PassionateKitten.sol#L553
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `burnFrom` function. As a result, the targeted address will lose the corresponding tokens.

```
function burnFrom(
    address from,
    uint256 amount
  ) public override onlyOwner whenNotPaused {
    if (!isBurnable()) {
      revert BurningNotEnabled();
    }
    super.burnFrom(from, 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

CR - Code Repetition

Criticality	Minor / Informative
Location	PassionateKitten.sol#L417,460
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.

```
 function transfer(
    address to,
    uint256 amount
  )
    public
    virtual
    override
    whenNotPaused
    validateTransfer(msg.sender, to)
    returns (bool)
  {
    uint256 taxAmount = _taxAmount(msg.sender, amount);
    uint256 deflationAmount = _deflationAmount(amount);
    uint256 amountToTransfer = amount - taxAmount -
deflationAmount;

    if (isMaxAmountOfTokensSet()) {
      if (balanceOf(to) + amountToTransfer >
maxTokenAmountPerAddress) {
        revert DestBalanceExceedsMaxAllowed(to);
      }
    }

    if (taxAmount != 0) {
      _transfer(msg.sender, taxAddress, taxAmount);
    }
    if (deflationAmount != 0) {
      _burn(msg.sender, deflationAmount);
    }
    return super.transfer(to, amountToTransfer);
  }
```

```
function transferFrom(
    address from,
    address to,
    uint256 amount
  )
    public
    virtual
    override
    whenNotPaused
    validateTransfer(from, to)
    returns (bool)
  {
    uint256 taxAmount = _taxAmount(from, amount);
    uint256 deflationAmount = _deflationAmount(amount);
    uint256 amountToTransfer = amount - taxAmount -
deflationAmount;

    if (isMaxAmountOfTokensSet()) {
      if (balanceOf(to) + amountToTransfer >
maxTokenAmountPerAddress) {
        revert DestBalanceExceedsMaxAllowed(to);
      }
    }

    if (taxAmount != 0) {
      _transfer(from, taxAddress, taxAmount);
    }
    if (deflationAmount != 0) {
      _burn(from, deflationAmount);
    }

    if (isForceTransferAllowed() && owner() == msg.sender) {
      _transfer(from, to, amountToTransfer);
      return true;
    } else {
      return super.transferFrom(from, to, amountToTransfer);
    }
  }
```

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, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	PassionateKitten.sol#L293,508,538,553,591,601
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function setDocumentUri(
string memory newDocumentUri
) external onlyOwner whenNotPaused {}
function mint(address to, uint256 amount) external onlyOwner
whenNotPaused {}
function burn(uint256 amount) public override onlyOwner
whenNotPaused {}
function burnFrom(
address from,
uint256 amount
) public override onlyOwner whenNotPaused {}
function renounceOwnership() public override onlyOwner
whenNotPaused {}
function transferOwnership(
address newOwner
) public override onlyOwner whenNotPaused {}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

RF - Redundant Functionalities

Criticality	Minor / Informative
Location	PassionateKitten.sol#L284,308,331,355,372,392,567,578,613,631,647,662
Status	Unresolved

Description

The contract includes functionalities to support features such as wallet size restrictions, taxation, deflation, whitelisting, and pausing. However, these functionalities are never executed because the respective control variables are set to `false` and cannot be modified.

```
function getWhitelistedAddresses() external view returns (address[]
memory) {}
function setMaxTokenAmountPerAddress(
uint256 newMaxTokenAmount
) external onlyOwner whenNotPaused {}
function blackList(address addr) external onlyOwner whenNotPaused
{}
function removeFromBlacklist(address addr) external onlyOwner
whenNotPaused {}
function setTaxConfig(
address _taxAddress,
uint256 _taxBPS
) external onlyOwner whenNotPaused {}
function setDeflationConfig(
uint256 _deflationBPS
) external onlyOwner whenNotPaused {}
function pause() external onlyOwner {}
function unpause() external onlyOwner {}
function updateWhitelist(
address[] calldata updatedAddresses
) external onlyOwner whenNotPaused {}
function _addManyToWhitelist(address[] calldata addresses) private
{}
function _clearWhitelist() private {}
function _taxAmount(
address sender,
uint256 amount
) internal view returns (uint256 taxAmount) {}
```

Recommendation

Removing redundant or unreachable code improves readability, optimizes gas usage, and enhances overall code consistency. The team is advised to remove the respective segments.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PassionateKitten.sol#L373,374,393
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.

```
address _taxAddress
uint256 _taxBPS
uint256 _deflationBPS
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PassionateKitten.sol#L181,195,383
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.

```
taxAddress = _taxAddress
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.

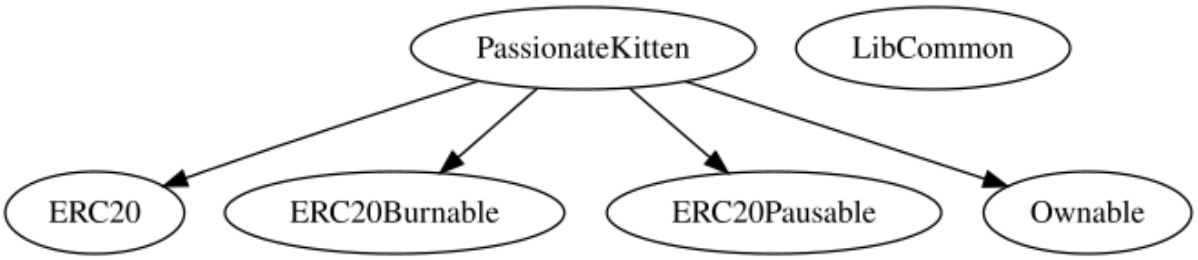
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
PassionateKitten	Implementation	ERC20, ERC20Burnable, ERC20Pausable, Ownable		
		Public	✓	ERC20
	_beforeTokenTransfer	Internal	✓	
	isPausable	Public		-
	isMintable	Public		-
	isBurnable	Public		-
	isBlacklistEnabled	Public		-
	isWhitelistEnabled	Public		-
	isMaxAmountOfTokensSet	Public		-
	isDocumentUriAllowed	Public		-
	isForceTransferAllowed	Public		-
	decimals	Public		-
	isTaxable	Public		-
	isDeflationary	Public		-
	getWhitelistedAddresses	External		-
	setDocumentUri	External	✓	onlyOwner whenNotPaused
	setMaxTokenAmountPerAddress	External	✓	onlyOwner whenNotPaused

	blackList	External	✓	onlyOwner whenNotPaused
	removeFromBlacklist	External	✓	onlyOwner whenNotPaused
	setTaxConfig	External	✓	onlyOwner whenNotPaused
	setDeflationConfig	External	✓	onlyOwner whenNotPaused
	transfer	Public	✓	whenNotPaused validateTransfer
	transferFrom	Public	✓	whenNotPaused validateTransfer
	mint	External	✓	onlyOwner whenNotPaused
	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 whenNotPaused
	_addManyToWhitelist	Private	✓	
	_clearWhitelist	Private	✓	
	_taxAmount	Internal		

	_deflationAmount	Internal		
LibCommon	Library			
	safeTransferETH	Internal	✓	
	validateAddress	Internal		
	safeTransferFrom	Internal	✓	
	safeTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
PassionateKitten
isDocumentUriAllowed
setDocumentUri
DocumentUriNotAllowed
MaxTokenAmountPerAddrLtPrevious
setMaxTokenAmountPerAddress
MaxTokenAmountNotAllowed
_mint
InvalidDecimals
InvalidMaxTokenAmount
transferOwnership
<Constructor>
InvalidTaxBPS
InvalidDeflationBPS
decimals
getWhitelistedAddresses
isTaxable
setTaxConfig
TokenIsNotTaxable
setDeflationConfig
isDeflationary
TokenIsNotDeflationary
WhitelistNotEnabled
updateWhitelist
addManyToWhitelist
CannotWhitelistBlacklistedAddr
_clearWhiteList
AddrAlreadyBlacklisted
CannotBlacklistWhitelistedAddr
blackList
BlacklistNotEnabled
isWhitelistEnabled
removeFromBlacklist
AddrAlreadyUnblacklisted
isBlacklistEnabled
validateTransfer
SenderNotWhitelisted
SenderBlacklisted
RecipientNotWhitelisted
RecipientBlacklisted
owner
mint
isMintable
MintingNotEnabled
isMaxAmountOfTokensSet
balanceOf
DestBalanceExceedsMaxAllowed
isForceTransferAllowed
transferFrom
_transfer
transfer
_taxAmount
_deflationAmount
_burn
burn
BurningNotEnabled
burnFrom
isBurnable
pause
_pause
unpause
PausingNotEnabled
renounceOwnership
isPausable
_beforeTokenTransfer
_unpause
LibCommon (lib)
validateAddress
safeTransferETH
safeTransferFrom
TransferFailed
safeTransfer
tokenAddress
call
Ownable
transferOwnership
renounceOwnership
_beforeTokenTransfer
transfer
transferFrom
mint
burn
burnFrom

Summary

Passionate kitten 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 burn tokens from any address. If the contract owner abuses the mint functionality, then the contract will be highly inflated. If the contract owner abuses the burn 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 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

cyberscope.io