Cyberscope

A TAC Security Company

Audit Report

PIXEL PUFFIN

June 2025

Network	BSC
Address	0x6Db95fb379f440113E383BE98AA6Dc57EBaDB2eb
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
Critical	AME	Address Manipulation Exploit	Unresolved
Minor / Informative	AAO	Accumulated Amount Overflow	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	MSC	Missing Sanity Check	Unresolved
Minor / Informative	NWES	Nonconformity with ERC-20 Standard	Unresolved
Minor / Informative	PAMAR	Pair Address Max Amount Restriction	Unresolved
Minor / Informative	POSD	Potential Oracle Stale Data	Unresolved
Minor / Informative	PTR	Potential Token Remnants	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L14	Uninitialized Variables in Local Scope	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved

●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved
●	L22	Potential Locked Ether	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	PixelPuffin
Compiler Version	v0.8.26+commit.8a97fa7a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6db95fb379f440113e383be98aa6dc57ebadb2eb
Address	0x6db95fb379f440113e383be98aa6dc57ebadb2eb
Network	BSC
Symbol	PIXP
Decimals	18
Total Supply	500,000,000,000
Badge Eligibility	Must Fix Criticals

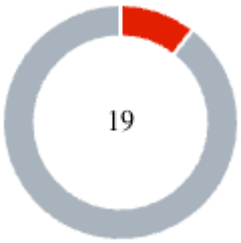
Audit Updates

Initial Audit	14 Jun 2025

Source Files

Filename	SHA256
PixelPuffin.sol	169b139b949073fdf63ac684b012ef6fc14ccb3225077feefc4b31d39df29a3f

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

The contract's pair address is not excluded from fees. Hence, sale transactions could be interrupted because of the max wallet restriction, as described in detail in section `PAMAR`. As a result, the contract might operate as a honeypot.

Recommendation

The team is advised to follow the recommendations outlined in the PAMAR finding and implement the necessary steps to mitigate the identified risk, ensuring that the contract does not operate as a honeypot.

AME - Address Manipulation Exploit

Criticality	Critical
Location	PixelPuffin.sol#L1415
Status	Unresolved

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts. Such vulnerabilities could enable attackers to exploit these functions, potentially leading to unauthorized actions or the execution of malicious code under the guise of legitimate operations.

The `burnLPTokens` function allows any external caller to burn LP tokens and receive a fixed BNB reward each time the function is called. However, the function does not validate whether the provided `lpTokenAddress` corresponds to a legitimate or approved LP token. A malicious actor can deploy a custom ERC20 token, mint arbitrary amounts, transfer ≥100 tokens to the contract, and repeatedly call `burnLPTokens`. Each call results in a BNB reward being sent to the caller, enabling them to drain the `accumulatedWBNB` balance without providing actual value.

```
function burnLPTokens(address lpTokenAddress) external nonReentrant {
    require(lpTokenAddress != address(0), "Zero address");
    uint256 lpBalance = IERC20(lpTokenAddress).balanceOf(address(this));
    require(lpBalance > 100 * 10**18, "LP tokens must be > 100");
    uint256 reward = (accumulatedWBNB * 1) / 1000;
    if (reward > 0) {
        accumulatedWBNB -= reward;
        IERC20(WBNB).safeTransfer(msg.sender, reward);
    }
    IERC20(lpTokenAddress).transfer(address(0xdead), lpBalance);
    emit LiquidityBurned(lpBalance);
}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. This could include checks against a whitelist of approved addresses, verification that the address implements a specific contract interface or other methods that confirm the legitimacy and integrity of the external contract. Implementing such validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1349
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
accumulatedFees += accumulationAmount;
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1265,1266
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
maxTxAmount
maxWalletAmount
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1273,1498,1503,1508,1513,1518
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
_isExcludedFromFees[account] = excluded;
PANCAKE_ROUTER = _router;
PANCAKE_V2_ROUTER = _routerV2;
WBNB = _wbnb;
CHAINLINK_BTCUSD = _priceFeed;
BTC_ORACLE = _oracle;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1523
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The contract does not check if the `_feeTier` is one of tiers supported by the router.

```
function updateFeeTier(uint24 _feeTier) external onlyDeployer {
    require(_feeTier > 0, "Invalid fee tier");
    FEE_TIER = _feeTier;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1323
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
require(amount > 0, "Transfer amount zero");
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1315
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
if (!_isExcludedFromFees[from] && !_isExcludedFromFees[to]) {
    require(amount <= maxTxAmount, "Transfer exceeds max transaction limit");
    require(balanceOf(to) + amount <= maxWalletAmount, "Wallet exceeds max wallet
limit");
}
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

POSD - Potential Oracle Stale Data

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1295
Status	Unresolved

Description

The contract relies on retrieving price data from an oracle. However, it lacks proper checks to ensure the data is not stale. The absence of these checks can result in outdated price data being trusted, potentially leading to significant financial inaccuracies.

```
function viewBTCSupply() external view returns (uint256) {
    uint256 btcMarketCap = IBTCOracle(BTC_ORACLE).getBTCMcap();
    (, int256 price, , ,) = IChainlinkPriceFeed(CHAINLINK_BTCUSD).latestRoundData();
    require(price > 0, "Invalid BTC price");

    uint256 btcPrice = uint256(price);
    uint256 adjustedBtcPrice = btcPrice * 10**10;

    return btcMarketCap / adjustedBtcPrice;
}
```

Recommendation

To mitigate the risk of using stale data, it is recommended to implement checks on the round and period values returned by the oracle's data retrieval function. The value indicating the most recent round or version of the data should confirm that the data is current. Additionally, the time at which the data was last updated should be checked against the current interval to ensure the data is fresh. For example, consider defining a threshold value, where if the difference between the current period and the data's last update period exceeds this threshold, the data should be considered stale and discarded, raising an appropriate error.

For contracts deployed on Layer-2 solutions, an additional check should be added to verify the sequencer's uptime. This involves integrating a boolean check to confirm the sequencer is operational before utilizing oracle data. This ensures that during sequencer downtimes,

any transactions relying on oracle data are reverted, preventing the use of outdated and potentially harmful data.

By incorporating these checks, the smart contract can ensure the reliability and accuracy of the price data it uses, safeguarding against potential financial discrepancies and enhancing overall security.

PTR - Potential Token Remnants

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1409,1410
Status	Unresolved

Description

The `addLiquidity` function resets `accumulatedFees` and `accumulatedWBNB` to zero after adding liquidity. However, the function logic doesn't guarantee that all tokens and WBNB are used during the `addLiquidity` call. Specifically, slippage during the `addLiquidity` call to the router can result in leftover tokens or WBNB remaining in the contract. Resetting `accumulatedFees` and `accumulatedWBNB` to zero in this scenario leads to a discrepancy between the tracked accumulated amounts and the actual token balances held by the contract, potentially causing accounting errors or unexpected behavior in subsequent operations that rely on these variables.

```
accumulatedFees = 0;
accumulatedWBNB = 0;
```

Recommendation

Before resetting `accumulatedFees` and `accumulatedWBNB`, reconcile the contract's token and WBNB balances with the amounts used for liquidity. Calculate the difference and adjust `accumulatedFees` and `accumulatedWBNB` accordingly to reflect the actual amounts remaining in the contract. This ensures accurate tracking of accumulated fees and WBNB, preventing potential accounting inconsistencies.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1248,1249,1250,1251,1252,1253,1496,1501,1506,1511,1516,1521
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 public PANCAKE_ROUTER;
address public PANCAKE_V2_ROUTER;
address public WBNB;
address public CHAINLINK_BTCUSD;
address public BTC_ORACLE;
uint24 public FEE_TIER;
...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1523
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.

```
FEE_TIER = _feeTier;
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PixelPuffin.sol#L35,783,1141,1151,1161
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 _contextSuffixLength() internal view virtual returns (uint256) {
        return 0;
    }

_burn(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidSender(address(0));
        }
        _update(account, address(0), value);
    }

    /

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1331,1332,1333
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 burnAmount;
uint256 accumulationAmount;
uint256 deployerAmount;
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1037,1066
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 ("memory-safe") {
            let success := call(gas(), token, 0, add(data, 0x20), mload(data), 0,
0x20)
            // bubble errors
            if iszero(success) {
                let ptr := mload(0x40)
                returndatacopy(ptr, 0, returndatasize())
                revert(ptr, returndatasize())
            }
            returnSize := returndatasize()
            returnValue := mload(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	PixelPuffin.sol#L14,46,148,177,185,268,276,363,529,557,867,1081,1170
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 ^0.8.20;
pragma solidity ^0.8.20;
pragma solidity ^0.8.25;
```

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	PixelPuffin.sol#L46
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 ^0.8.20;
pragma solidity ^0.8.25;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1427,1491
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(lpTokenAddress).transfer(address(0xdead), lpBalance);
IERC20(tokenAddress).transfer(deployer, amount);
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	PixelPuffin.sol#L1533
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {
        revert("BNB not accepted");
    }
```

Recommendation

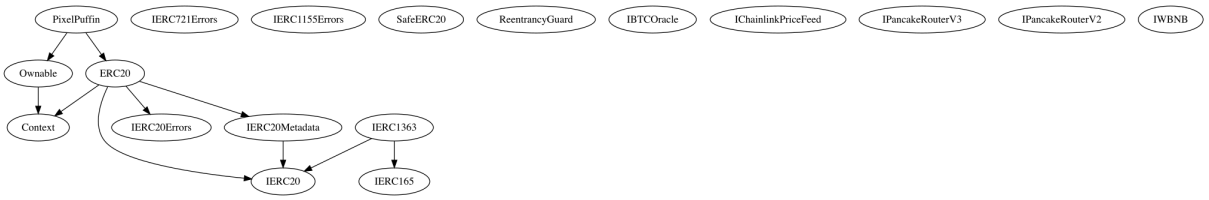
The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

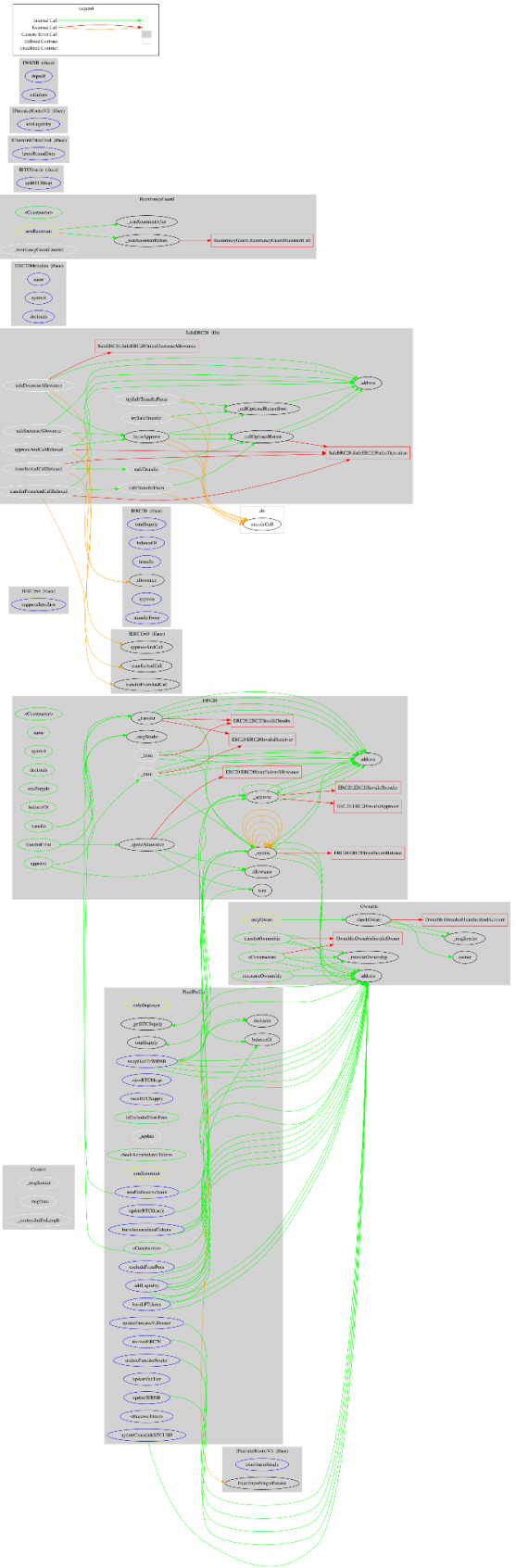
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
PixelPuffin	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	excludeFromFees	External	✓	onlyOwner
	_getBTCSupply	Internal		
	viewBTCMcap	External		-
	viewBTCSupply	External		-
	isExcludedFromFees	Public		-
	_update	Internal	✓	
	checkAccumulatedTokens	Public		-
	addLiquidity	External	✓	nonReentrant
	burnLPTokens	External	✓	nonReentrant
	burnAccumulatedTokens	External	✓	nonReentrant
	swapHalfToWBNB	External	✓	nonReentrant
	recoverERC20	External	✓	-
	updatePancakeRouter	External	✓	onlyDeployer
	updatePancakeV2Router	External	✓	onlyDeployer
	updateWBNB	External	✓	onlyDeployer
	updateChainlinkBTCUSD	External	✓	onlyDeployer
	updateBTCOracle	External	✓	onlyDeployer
	updateFeeTier	External	✓	onlyDeployer

	sendToReserveVault	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

PIXEL PUFFIN contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. PIXEL PUFFIN is an interesting project that has a friendly and growing community. The Smart Contract analysis reported some critical issues that require the contract’s redeployment to be fixed. The fees are locked at 5%.

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 TAC 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.

A **TAC Security** Company

The Cyberscope team

cyberscope.io